

# OPEN\_NEXT

## Deliverable 3.2

### Annexure: Platform Demonstrator for collaborative engineering



This project is funded by the European Union’s Horizon 2020 Research and Innovation Programme under the Grand Agreement no. 869984.



## OPEN\_NEXT – Transforming collaborative product creation

### Consortium:

#	Participant Legal Name	Short Name	Country
1	TECHNISCHE UNIVERSITAT BERLIN	TUB	DE
2	INSTITUT POLYTECHNIQUE DE GRENOBLE	GINP	FR
3	ALEXANDER VON HUMBOLDT-INSTITUT FURINTERNET UND GESELLSCHAFT GGMBH	HIIG	DE
4	UNIVERSITY OF BATH	UBA	UK
5	ZENTRUM FUR SOZIALE INNOVATION GMBH	ZSI	AT
6	FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V.	FHG	DE
7	DANSK DESIGN CENTER APS	DDC	DK
8	WIKIMEDIA DEUTSCHLAND - GESELLSCHAFT ZUR FÖRDERUNG FREIEN WISSENS EV	WMDE	DE
9	WIKIFACTORY EUROPE SL	WIF	ES
10	STICHTING WAAG SOCIETY	WAAG	NL
11	MAKER	MAK	DK
12	AGILE HEAP EV	FLB	DE
13	SONO MOTORS GMBH	SOM	DE
14	OPNTEC GMBH	OPT	DE
15	STYKKA APS	STY	DK
16	TILL WOLFER	XYZC	DE
17	FICTION FACTORY	FIF	NL
18	M2M4ALL	SOD	NL
19	INNOC OSTERREICHISCHE GESELLSCHAFT FUR INNOVATIVE COMPUTERWISSENSCHAFTEN	HAL	AT

Duration: 09/2019-08/2022

Grant: H2020-869984

Contact (coordinator): Prof Dr-Ing Roland JOCHEM

Address: Technische Universität Berlin, Sekretariat PTZ 3, Pascalstr. 8-9, 10587 Berlin

E-mail: [roland.jochem@tu-berlin.de](mailto:roland.jochem@tu-berlin.de)

**Disclaimer:** The contents of this document are not intended to replace consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at his/her sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the views of the author(s).

## D3.2 – “Annexure: Platform demonstrator for collaborative engineering”

Review and approval status:

	Name and Surname	Role in the Project	Partner
<b>Author(s)</b>	Sonika Gogineni, Martin Häuer	Work Package lead, Research Assistant	FHG
	Erik Paul Konietzko, Cansu Tanrikulu	Work package development, Student Assistant	FHG
	Max Kanpik, Diego Vaquero, Andrés Barreiro	Work package development	WIF
<b>Reviewed by</b>	Mehera Hassan	Research Associate	TUB
	Pen-Yuan Hsing	Postdoctoral Researcher	UBA
<b>Approved by</b>	Robert Mies	Project coordinator	TUB

History of changes:

Version	Date	Description of changes	By
0.1	21.06.2021	Initial draft	Sonika Gogineni
0.2	30.07.2021	Draft for review	Sonika Gogineni
1.0	26.08.2021	Final document for final project management verification	Sonika Gogineni, Erik Konietzko

Details:

<b>Dissemination level</b>	Open Access
<b>Due date</b>	31.08.2021
<b>Issue date</b>	31.08.2021
<b>Contract No.</b>	869984
<b>Responsible Partner</b>	FHG
<b>File name</b>	D3.2_Annexure_Platform Demonstrator for collaborative engineering.docx

Keywords:

Open source hardware, ICT for OSH, open source development, community management, skill ontology, interoperability, collaborative production, guidelines, and documentation

## Abstract:

This is a supporting document, which provides details about why and how the solutions were selected and developed. In addition, the solutions are enhanced with design notes and installation guide. The design notes provide insights into the structure of the solutions, and the installation guides provides a step-by-step description on how to install and use the solutions.

The solutions are the various demonstrators and software tools developed to support company-community collaboration in Open Source Hardware projects. Fraunhofer IPK and Wikifactory have carried out the developments. Based on the prioritized needs of the target group a set of concepts (referred to as solutions in the deliverable) were developed to address the needs. This led to the development of solutions under four main categories namely: community management, interoperability, collaborative production, and documentation and guidelines. The target groups identified for the solutions are companies, communities and platform owners/ creators.

Under community management, the focus was on matchmaking the company and community with skills and interests as the common connector. A first version of the solution as a demonstrator integrated in Wikifactory is presented in this deliverable.

Under the category interoperability, a stable version of an import-export tool was developed to facilitate transfer of files from one platform to another. Hence, providing the community and company to move the projects across platforms without facing a vendor lock-in.

Under the category of collaborative production, in order to assist a company/community member looking for producers/prototype manufacturers a first prototype of a matchmaking tool was developed on Wikifactory. The first step was to identify production metadata required to ensure a seamless communication, followed by automatically identifying metadata in project text and then finding suitable collaborations.

Under the category documentation and guidelines, four different templates and guides were developed to assist companies and contributors.

All the results represent a first release, which need to be validated and further developed during the course of the project.

## Table of contents

<b>1. Introduction</b>	<b>9</b>
1.1. User story prioritization process	9
<b>2. Import-Export Tool</b>	<b>12</b>
2.1. Development method	12
2.2. Design notes	12
2.2.1. Management of authentication tokens	14
<b>3. Skill ontology demonstrator</b>	<b>15</b>
3.1. Development method	15
3.1.1. User flows	15
3.1.2. Semantic network	17
3.2. Design notes	20
3.2.1. Class diagram	20
3.2.2. Flow instantiation	22
3.3. Installation	30
3.4. Ontologies	38
3.4.1. Resource Description Framework and Web Ontology Language	38
3.4.2. OWL documents	39
<b>4. Collaborative Production – Identifying production metadata</b>	<b>43</b>
4.1. Development method	43
4.2. Design notes	43
<b>5. Collaborative Production – Identifying production metadata in documentation through machine learning</b>	<b>45</b>
5.1. Development Method	45
5.2. Design notes	52
5.3. Installation guide	54
<b>6. Documentation and Guidelines – Project overview template</b>	<b>57</b>
6.1. Development method and design notes	57
<b>7. Documentation and Guidelines -Technology-readiness levels for OSH</b>	<b>57</b>
7.1. Development method and design notes	57
<b>8. Documentation and Guidelines - Licensing open source hardware guide</b>	<b>59</b>
8.1. Development method	59

### List of abbreviations and terms

API	Application programming interface
ESCO	European Skills/Competences, qualifications and Occupations
IDE	Integrated development environment
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
NASA	National Aeronautics and Space Administration
NER	Named Entity Recognition
NLP	Natural Language Processing
ODRL	Open Documentation Readiness Levels
OSD	Open source development
OSH	Open source hardware
OSS	Open source software
OTRL	Open Technology Readiness Levels
OWL	Web Ontology Language
RDF	Resource Description Framework
TRL	Technology Readiness Level
URL	Universal resource locator

### List of figures

Figure 1: Definitions with their relationships.....	8
Figure 2: Flowchart of the Import Export steps. In purple appear the high-level steps of the process. Success statuses appears in green while error status appears in yellow and red .....	13
Figure 3: Database log for a successful import-export job .....	13
Figure 4: Database log for an import-export job that required user authentication .....	14
Figure 5: Workshop with OPEN_NEXT project members to find and prioritize user flows .....	16
Figure 6: Ontology classes for the user story application .....	17
Figure 7: Collaborative process map (Mies 2021).....	18
Figure 8: Typological project types (Mies 2021) .....	18
Figure 9: Demonstrator class structure.....	20
Figure 10: Skill instantiation flow .....	23
Figure 11: Flowchart Instantiation (1/2) .....	25
Figure 12: Flowchart Instantiation (2/2) .....	26
Figure 13: Query execution flow .....	29
Figure 14: Coping the link of the GitHub repository page .....	31
Figure 15: Starting Eclipse and choosing a workspace.....	32
Figure 16: Setting the SSH2 key in the preferences .....	32
Figure 17: Open Git perspective in Eclipse (1/2).....	33
Figure 18: Open Git perspective in Eclipse (2/2).....	33
Figure 19: Clone a Git repository from the Git repositories tab .....	33
Figure 20: Inserting the repository URI to clone .....	34
Figure 21: Selection the branch to clone .....	34
Figure 22: Choosing a local directory for the git repository.....	35
Figure 23: Importing existing Maven project.....	35
Figure 24: Choosing root directory.....	36
Figure 25: Imported project in the project explorer tab.....	36
Figure 26: Setting JAVA Compiler level .....	37

Figure 27: Agreeing to the project build ..... 37

Figure 28: JUnit import error..... 37

Figure 29: Adding JUnit library ..... 38

Figure 30: Instantiation example..... 39

Figure 31: Ontology header..... 40

Figure 32: Declaration axioms for (a) Class Project (b) object property has\_member (c) data proerty Project\_title ..... 40

Figure 33: Declaration axiom for (a) individual CAD\_file (b) annotation property wif\_issue\_1\_cmap 40

Figure 34: Axioms to set (a) subclass relation and (b) disjointness of classes ..... 40

Figure 35: Classifying an individual ..... 40

Figure 36: Defining inverse properties ..... 41

Figure 37: Axioms defining the (a) domain and (b) range of a property..... 41

Figure 38: Defining the annotation property wif\_issue\_1\_cmap ..... 41

Figure 39: Property restriction on class Creator ..... 41

Figure 40: Provision of production metadata by publishing a plain text file („manifest file“) in a git repository ..... 44

Figure 41: Example of the training data ..... 46

Figure 42: Login for doccano ..... 47

Figure 43: Creating a project in doccano ..... 48

Figure 44: Importing a dataset for doccano..... 48

Figure 45: Creating labels in doccano ..... 49

Figure 46: Imported dataset in doccano ..... 50

Figure 47: Reading labelled dataset ..... 50

Figure 48: Converting the format..... 51

Figure 49: Training the model ..... 51

Figure 50: Iterations through training..... 52

Figure 51: Saving the model ..... 52

Figure 52: Testing the model..... 52

Figure 53: Flowchart of steps to recreate a NER..... 53

Figure 54: Copying the URL of the repository ..... 54

Figure 55: Create Git Repository ..... 54

Figure 56: Clone the URL..... 55

Figure 57: The cloned repository in IDE ..... 55

Figure 58: Front end of the application..... 56

### List of definitions

- Company:** An organisation or entity
- Community:** A community is a group of people or organisations. They can take up different roles in projects. Some of them works on projects online using online platforms to carry out certain tasks (Dai et al.).
- Community member:** Community member is described as anyone who is interested in projects or other users for example on an online platform. They can duplicate the project without any contribution for their own purpose. (Li und Seering).
- Project team:** An open source project team consists of the core team and contributors (Dai et al.)
- Core team:** Core team usually builds and manages the backbone of project, they usually are responsible for major decisions. They are a part of the project team.
- Contributor:** From community members, the ones who are interested and contributing in a project, called contributors (Dai et al.)

The above definitions are further illustrated in Figure 1.

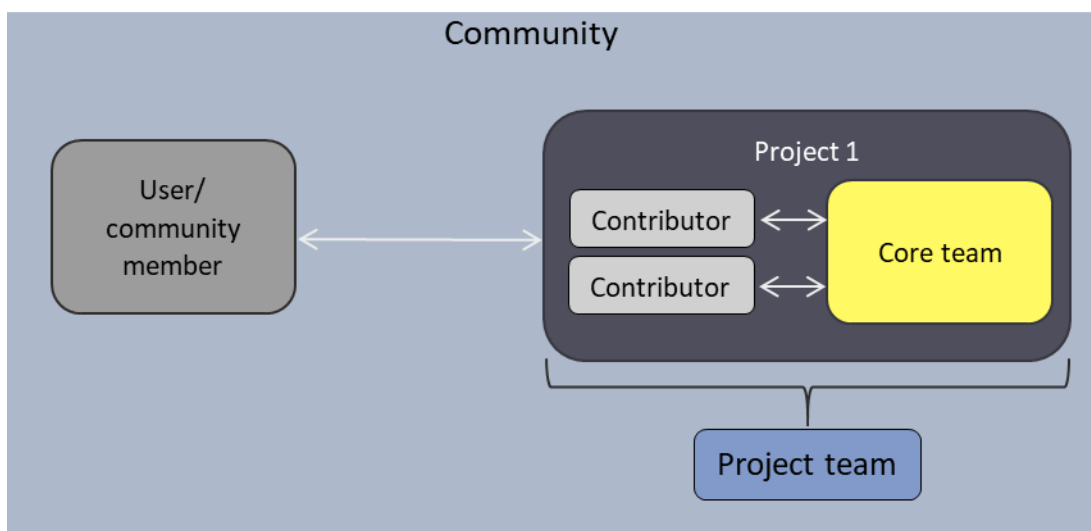


Figure 1: Definitions with their relationships



# 1. Introduction

This document acts as a supporting annexure to the main document (D3.2\_ Platform demonstrator - Collaborative engineering platform demonstrator). The document mainly focuses on detailing the methodology used to develop the solutions, supporting design notes and installation guides.

Following the results of the previous deliverable D3.1, there was a need to prioritize the needs/user stories identified. Hence, a prioritization process was carried out. The methodology used to do this is detailed in the following section.

## 1.1. User story prioritization process

For the prioritization of the user story groups the weighted average was calculated, taking the quantities of mentions and user stories into account (cf. Equation 1). The average was calculated for both within and among the groups, and for each user story individually, in order to provide as heterogeneous a picture as possible.

Equation 1

$$x_{w,i} = \frac{w_{i,user}x_{i,user} + w_{i,mentions}x_{i,mentions}}{w_{i,user} + w_{i,mentions}}$$

- $x_{i,w}$  = weighted average of user story group i
- $w_i$  = weight factor (of user story or mentions) in user story group i
- $x_i$  = realisation of quantity (of user story or mentions) in user story group i

Table 1: User story group prioritisation statistics

Group #	User story group	# user stories	# mentions	% user stories	% mentions	Summary	Ranking
1	Community management	9	66	11.25%	25.38%	18.32%	1
2	Company community management	2	2	2.50%	0.77%	1.63%	
3	Documentation	9	25	11.25%	9.62%	10.43%	4
4	Informal Knowledge documentation	2	2	2.50%	0.77%	1.63%	
5	Interoperability	3	13	3.75%	5.00%	4.38%	
6	Collaborative Production	5	14	6.25%	5.38%	5.82%	
7	Status	1	9	1.25%	3.46%	2.36%	
8	Quality	2	4	2.50%	1.54%	2.02%	
9	Business Model	2	8	2.50%	3.08%	2.79%	
10	Certification	3	3	3.75%	1.15%	2.45%	
11	Liability	1	3	1.25%	1.15%	1.20%	
12	Guidelines	11	38	13.75%	14.62%	14.18%	2
13	Collaborative Product development	8	22	10.00%	8.46%	9.23%	
14	Development platform	10	26	12.50%	10.00%	11.25%	3
15	Collaborative project management	1	2	1.25%	0.77%	1.01%	
16	Community maintained project	1	3	1.25%	1.15%	1.20%	
17	Fablab/Makerspace	5	10	6.25%	3.85%	5.05%	
18	Discoverability	1	4	1.25%	1.54%	1.39%	
19	Collaborative Testing	2	4	2.50%	1.54%	2.02%	
20	Standards	2	2	2.50%	0.77%	1.63%	

	Sum	80	260	100%	100.00%	100.00%	
Weight user story	1						75% quantile
Weight mentions	1			+			50% quantile

This method was selected to contrast and level the influence of many, but less-mentioned user stories in a group with the influence of few, but often-mentioned user stories in a group. The following classification was selected as a gradation of the user stories in the evaluation:

- high evaluation: make up the area above the top quartile in an ordered frequency table,
- medium evaluation: make up the area between the middle and top quartile in an ordered frequency table.

Areas below the middle quartile were not considered further due to the high number of user stories for the first iteration of the selection. The average was calculated both within and among the groups, and for each user story alone. It turned out that the weighting (number of mentions vs. number of user stories) had a rather small influence in the area of the groups with high priority anyway, which confirmed that groups that were often mentioned (in the table e.g. community management) also contained the user stories with high mentions. This insight simplified the selection process based on the pure numbers of the surveys. Based on these, user story groups with high and medium ratings were selected for the further path. These are marked in the table with green (high) and yellow (medium).

The results from WP2 from the interviews conducted from the GINP was also assessed and mapped to the user stories and the categories. The ranking from above analysis and the results from the analysis from WP2 are compared in Table 2.

Table 2: User story group prioritization in comparison with interview results from WP2

Group #	User story group	# user stories	# mentions	Ranking	# needs from WP2	Ranking WP2
1	Community management	9	66	1	15	4
2	Company community management	2	2		1	
3	Documentation	9	25	4	53	1
4	Informal Knowledge documentation	2	2		3	
5	Interoperability	3	13		2	
6	Collaborative Production	5	14		3	
7	Status	1	9		5	
8	Quality	2	4		4	
9	Business Model	2	8		2	
10	Certification	3	3		2	
11	Liability	1	3		1	
12	Guidelines	11	38	2	16	3
13	Collaborative Product development	8	22		9	
14	Development platform	10	26	3	24	2
15	Collaborative project management	1	2		1	
16	Community maintained project	1	3		1	
17	Fablab/Makerspace	5	10		5	
18	Discoverability	1	4		2	
19	Collaborative Testing	2	4		2	
20	Standards	2	2		2	

The number of needs from WP2 was calculated by assigning the identified needs from GINP into the 20 categories. The table indicates that the top 4 categories with prioritized needs are community management, documentation, guidelines and development platform. The focus of the interviews from GINP was on reuse of documentation, hence this could have made the difference in the order of ranking among the four categories compared to the user stories ranking (from D3.1).

## 2. Import-Export Tool

### 2.1. Development method

The Import Export service has been developed based into three main principles. First, it must facilitate the operation of *importing* files from the *import service* (i.e., download them) and *exporting* those files to the *exporting service* (i.e., upload them). Second, it must be *flexible* enough to incorporate as many services as possible for both operations, importing and exporting. Third, it should be usable from the endpoint users as well as allowing its execution from external services.

With these three pillars it was decided to develop the Import Export tool as a *microservice*. Thus, the service has some benefits such as being easily and independently deployable, easy to extend and manipulate and keeping all the functionality gathered up.

### 2.2. Design notes

Figure 2 shows all the import export process in a flowchart. The initial point of the diagram is the “*Start I/E process*”, which represent the start of a new import-export job.

Once the *job* has started, the service tries to *import* (i.e., download) the files from the *import service*. There are two possibilities in this step, either the importer finds some error while downloading the files, or the whole import step finishes successfully.

If an error was found while importing, the service checks if this is the first time that the job has been executed or not. If this is the first iteration of the job, the service sets the job status as “*IMPORTING\_ERROR\_AUTHORIZATION\_REQUIRED*”, which indicates that it is waiting for the user to retry again the process but providing different credentials. Otherwise, when this is not the first iteration of the job, that “*AUTHORIZATION\_REQUIRED*” status will already be present in the database. By default, the service assumes that if the user provided credentials to access the *import\_url* and there was an error, either the user does not have access to those files, or the URL is not reachable. In that case, the “*IMPORTING\_STATUS\_DATA\_UNREACHABLE*” will be set.

After finished downloading the files, the status of the job will be “*IMPORTING\_SUCCESSFULLY*” and then, the exporting step begins which is analogous to the importing one but changing the status names to “*EXPORTING*”. Finally, once both, importing and exporting steps have finished, the whole job will be marked as finished by setting its status to “*FINISHED\_SUCCESSFULLY*”.

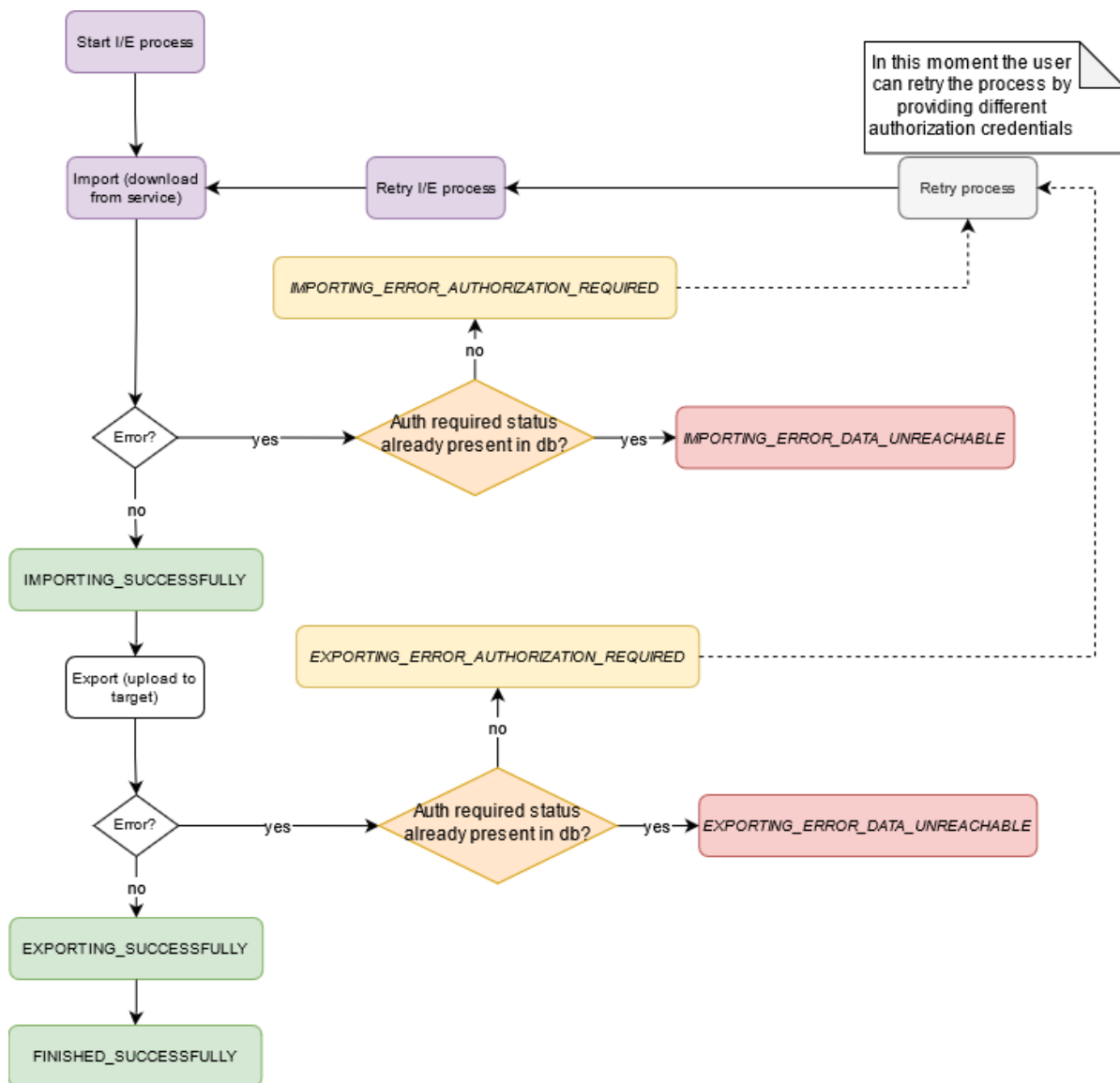


Figure 2: Flowchart of the Import Export steps. In purple appear the high-level steps of the process. Success statuses appears in green while error status appears in yellow and red

For example, Figure 3 shows the entries in the database for a successful execution of the import-export process.

	job_id	timestamp	from_status	to_status
1	e9902020-dc1d-4b73-b54d-d4e8653b418c	2021-07-13 13:39:28	[NULL]	PENDING
2	e9902020-dc1d-4b73-b54d-d4e8653b418c	2021-07-13 13:39:28	PENDING	IMPORTING
3	e9902020-dc1d-4b73-b54d-d4e8653b418c	2021-07-13 13:39:29	IMPORTING	IMPORTING_SUCCESSFULLY
4	e9902020-dc1d-4b73-b54d-d4e8653b418c	2021-07-13 13:39:29	IMPORTING_SUCCESSFULLY	EXPORTING
5	e9902020-dc1d-4b73-b54d-d4e8653b418c	2021-07-13 13:39:37	EXPORTING	EXPORTING_SUCCESSFULLY
6	e9902020-dc1d-4b73-b54d-d4e8653b418c	2021-07-13 13:39:38	EXPORTING_SUCCESSFULLY	FINISHED_SUCCESSFULLY

Figure 3: Database log for a successful import-export job

On the other hand, Figure 4 shows the database logs for a process in which the user had to authenticate to grant access to the files in the *import service*.

1695b176-9fed-48b0-b024-1343ccdc5e85	2021-07-13 13:43:46	[NULL]	PENDING
1695b176-9fed-48b0-b024-1343ccdc5e85	2021-07-13 13:43:46	PENDING	IMPORTING
1695b176-9fed-48b0-b024-1343ccdc5e85	2021-07-13 13:43:46	IMPORTING	IMPORTING_ERROR_AUTHORIZATION_REQUIRED
1695b176-9fed-48b0-b024-1343ccdc5e85	2021-07-13 13:44:22	IMPORTING_ERROR_AUTHORIZ	PENDING
1695b176-9fed-48b0-b024-1343ccdc5e85	2021-07-13 13:44:22	PENDING	IMPORTING
1695b176-9fed-48b0-b024-1343ccdc5e85	2021-07-13 13:44:24	IMPORTING	IMPORTING_SUCCESSFULLY
1695b176-9fed-48b0-b024-1343ccdc5e85	2021-07-13 13:44:24	IMPORTING_SUCCESSFULLY	EXPORTING
1695b176-9fed-48b0-b024-1343ccdc5e85	2021-07-13 13:44:25	EXPORTING	EXPORTING_SUCCESSFULLY
1695b176-9fed-48b0-b024-1343ccdc5e85	2021-07-13 13:44:25	EXPORTING_SUCCESSFULLY	FINISHED_SUCCESSFULLY

Figure 4: Database log for an import-export job that required user authentication

### 2.2.1. Management of authentication tokens

As a side note with regards to the design of the import-export service, it is important to mention the authentication flow for the *import* and *export* services. Thus, each HTTP REST request to the Import Export service may require an *import\_token* as well as an *export\_token* fields. Those fields will be used to grant access for the import-export service to either download files from or to upload to a certain service. In some cases, this field is not required and then it can be left empty such as in the case of importing a GitHub public repository which needs no authentication.

On the other hand, some platforms such as Google Drive or Dropbox requires the user to grant access to the Import Export service for downloading or uploading the files. Both services use the OAuth 2.0 authentication protocol. Thus, once the user has authenticated, each request to those services will be authorized. We now present the common steps for the OAuth 2.0 process to be considered for the Import Export tool.

1. The first step consists of creating an application in the developer area of the desired service. Depending on the service, this process can be done either from the Google API Console<sup>1</sup> or the Dropbox Developer page<sup>2</sup>. In the first case, once created a new project from the Google API console, it is mandatory to first enable the Google Drive API.
2. Create an Oauth key with the necessary permissions. In this case, the Import Export tool will need access for either downloading or uploading files, depending on the case.
3. When authentication and authorization is required, the user will see a consent screen, which will ask her to enter an account and confirm that she allows the Import Export service to act on her behalf.
4. Once the user has given the consent, she is redirected to the initial point and the OAuth token has been generated and can be integrated in the REST request.

<sup>1</sup> <https://console.cloud.google.com/apis/dashboard?pli=1>

<sup>2</sup> <https://www.dropbox.com/developers/apps>

### 3. Skill ontology demonstrator

#### 3.1. Development method

The main user stories for the focus of the demonstrator development are to find and attract right contributors, motivating them to contribute and to create a collaborative development environment. In order to be able to rely on proven and well accepted techniques for this purpose, it was analysed how contributors interact with projects in OSS development, since virtual development methods, documentation and communication are used to a large extent in OSH, too. Here, tagging has become established in the course of the last few years. To help contributors in different ways in OSS, software and social tagging became a popular mechanism to enrich descriptions with additional annotations, describe tasks or add any other information to an artefact (Treude and Storey 2009) (Wang et al. 2014) (Ding et al. 2009). This was used as inspiration to assign skills to the tasks and match them skill tags in individual profiles. For the sake of straightforwardness, simplicity and recognition for contributors and project teams, the approach based on tagging was chosen for the further development. The next step was to identify user flows (user interactions) for various situations are on platforms such as Wikifactory.

##### 3.1.1. User flows

The user flows of the skill matching application case are based on the selected user stories described in the previous section. In this endeavor, an analysis using the user journey<sup>3</sup> on the WIF platform was carried out, to identify possible implementation ideas and needs. Those were documented and the main commonalities identified, which revealed that in each case, skills must first be assigned to user profiles and the corresponding counterpart (e. g. issue or project). Based on this, the ideas and needs identified were transcribed into an evaluation template (Figure 5), where users, tasks and projects are connected and the possible implementations were stated from the different points of views:

1. a user, that provides and looks for enhancing their skills
2. a project in need of competences and searching for skilled contributors
3. tasks, that have to be enriched with information increasing expressiveness for people to choose or assign.

This template was presented to the OPEN\_NEXT project partners in a general assembly workshop to evaluate, weight and comment on the issues that can be implemented.

---

<sup>3</sup> [https://depositonce.tu-berlin.de/bitstream/11303/10962/6/User\\_journey\\_Wikifactory.pdf](https://depositonce.tu-berlin.de/bitstream/11303/10962/6/User_journey_Wikifactory.pdf)

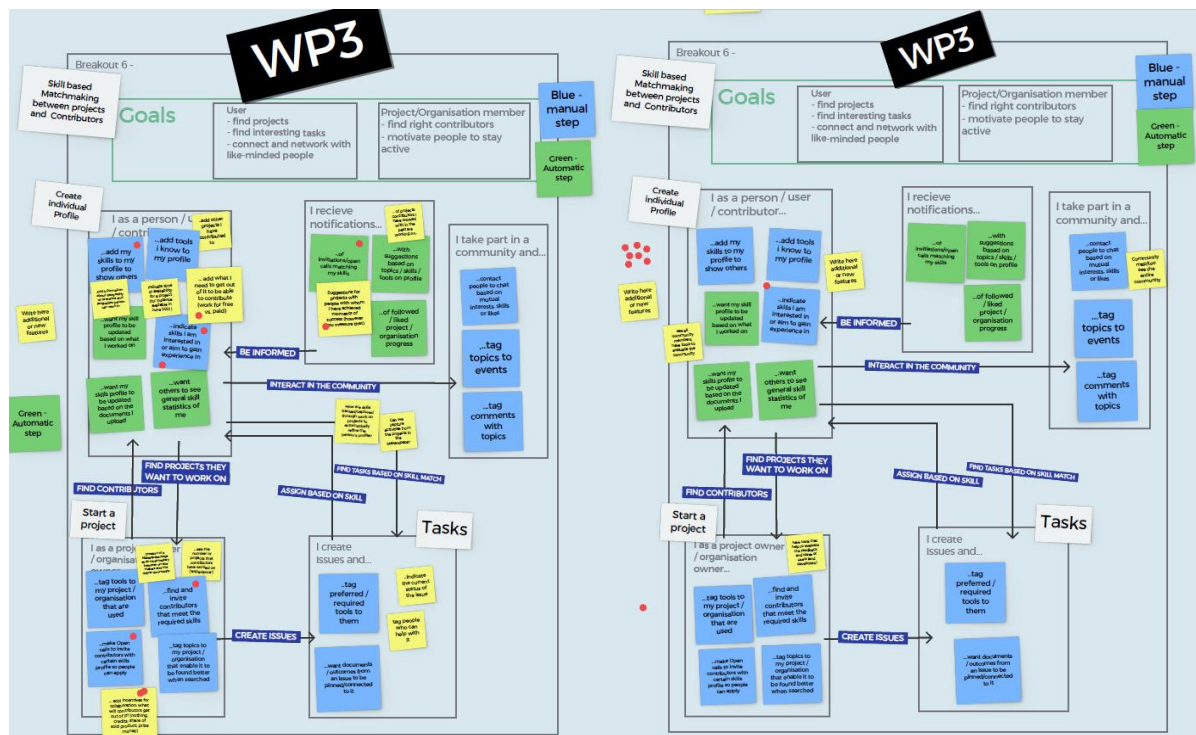


Figure 5: Workshop with OPEN\_NEXT project members to find and prioritize user flows

Workshop participants deemed this to be a useful solution and its focus on user friendliness was well accepted. It was also pointed out that it is helpful to be able to assess the capabilities of the users more easily and link them to projects and issues. This provides a good first approach to evaluating their skills.

Besides general approval, they raised concerns about the automation of a skill matching service and how this approach could be promoted to users to realize the benefits and use the feature, since of course at some point in the information flow a user has to manually assign some skills and interests to the user profile. This point was well received and is considered as question for the validation rounds with real user interaction.

From the user perspective two topics were raised to be considered during the further development. The first is to consider the incentives and motivation of the users during a matching process, e. g. to state what skills a user would offer in exchange for what services (meaning doing something for fun, for reputation, in exchange for money or help on another subject, etc.). The second, to consider the difficulty to virtually track skills and task fulfilment of physical tasks, e. g. building a prototype.



### 3.1.2. Semantic network

Based on the user flows a semantic network was developed. The net structure of the main classes of the semantic network for the application case are shown in Figure 6. For this purpose, the details of the semantic network are described and the classes specific to the user stories are dealt with in more detail. The ontology language used is OWL, which is one of the most used for semantic modelling language. OWL is an RDF based language and extends the RDF specification with additional predefined expressive relations (Pan et al. 2017).

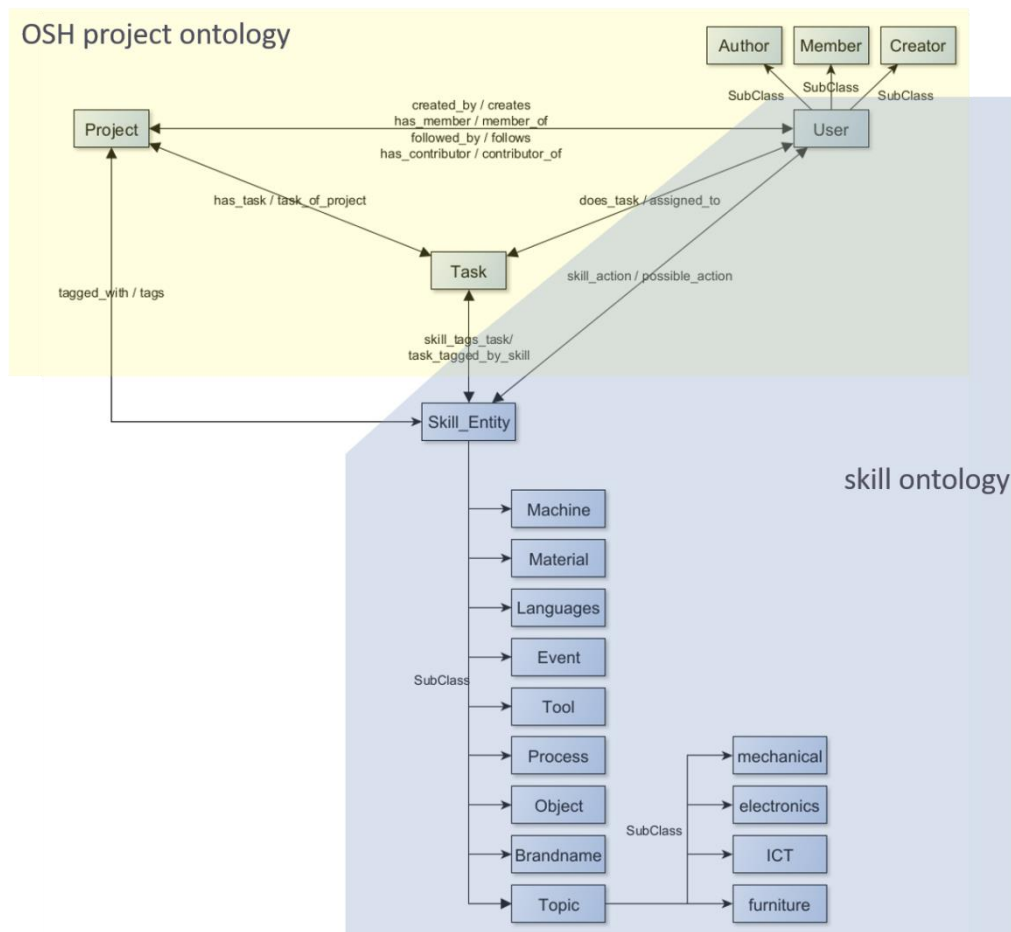


Figure 6: Ontology classes for the user story application

#### 1. OSH project ontology

The OSH Project ontology is inspired by the collaborative process map (CPM) (Mies 2021) developed in *OPEN!*<sup>4</sup> (Figure 7). The CPM depicts the work flow in a co-development environment of open communities. In particular, it is oriented towards the working methods of collaborative innovators. This group was selected for the creation of such a generic process based on a typology. The typology was done by classifying and comparing the criteria "Decentralization of contributions" and "Pursued level of collaborative development" of different project groups, whereby the collaborative innovators group was assigned high rankings in both (Figure 8). This shows not only the intention for decentralized collaboration, but also the actual implementation of this. The working method on the one hand and the most likely open documentation on the other were assessed as suitable for identifying procedural structures. Likewise, this way of working corresponds to a new terrain in development that contrasts

<sup>4</sup> [www.qw.tu-berlin.de/menue/forschung/projekte/abgeschlossene\\_projekte/open/](http://www.qw.tu-berlin.de/menue/forschung/projekte/abgeschlossene_projekte/open/)

with established processes in industrial design and therefore is worth exploring. The CPM artefacts used and created are oriented towards the development platform GitHub, which was selected as representative collaboration platform for OSH development. Relevant terms of the CPM and the relations between these have been analyzed and given appropriate attention in the OSH Project ontology, e. g. by converting them into ontology classes and properties. Explicit data properties in the ontology also resulted from the application data of the WIF platform GraphQL API.

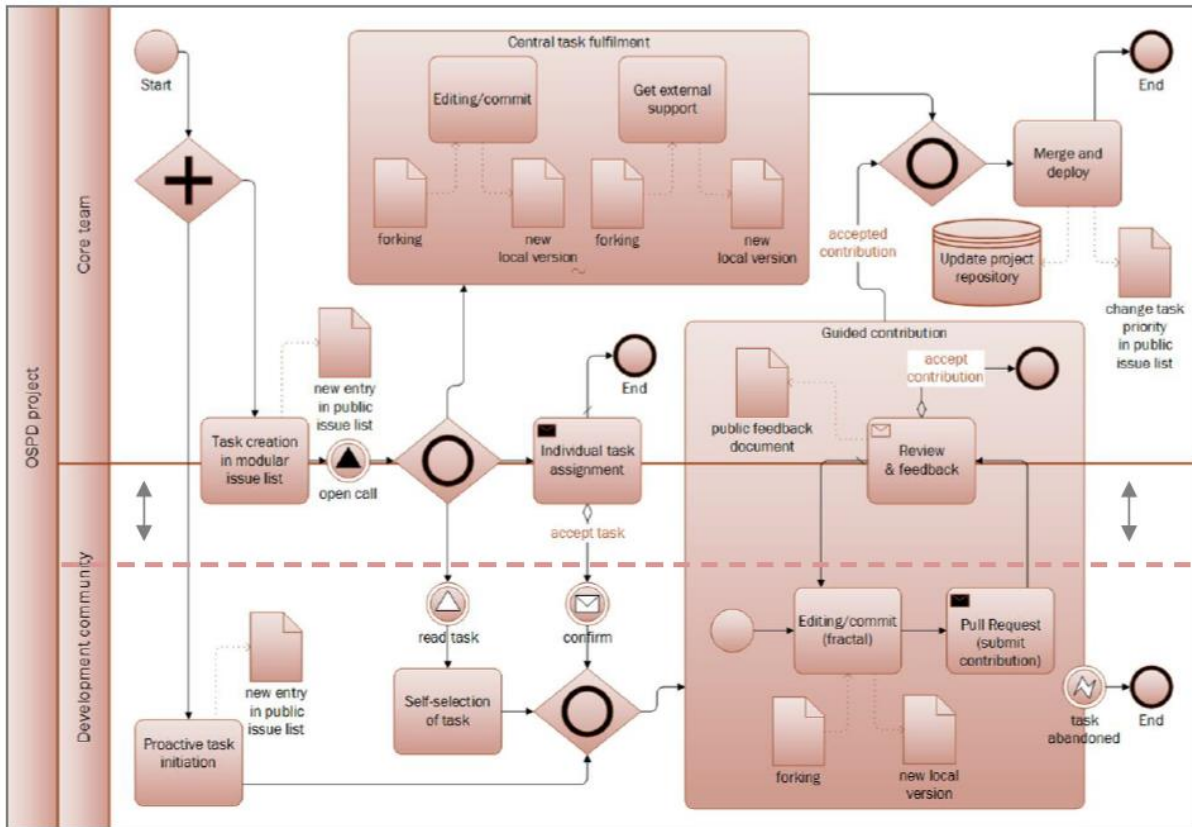


Figure 7: Collaborative process map (Mies 2021)

<b>Decentralisation of contributions</b>	high [3 – 4]	none	none	4x projects collaborative innovators
	intermediate [2 – 3]	none	none	2x projects
	Low [1 – 2]	public innovators 3x projects	4x projects	involuntary isolated innovators 7x projects
		low [1 – 2]	intermediate [2 – 3.5]	high [3.5 – 5]
		<b>Pursued level of collaborative development</b>		

Figure 8: Typological project types (Mies 2021)

## 2. Skill ontology

The skill ontology is based on terms from the European Skills/Competences, qualifications and Occupations (ESCO) hierarchy<sup>5</sup>. This hierarchy was analyzed and the skills were reduced for the project relevant topics of electronics, mechanics, ICT, furniture and cars/mobility. From the hierarchy of the remaining skills, a structure should emerge that is more general and suitable for the free and independent work in OSH. (The tagging of skills would need to be done by a user at some point; too many and specialized skills would cause confusion and be a barrier to use.) The main nouns and verbs of the skills were used to identify corresponding skill terminologies which occur in the OSH world. For this, a sample of project, user and issue data from the WIF platform was used, resulting in the identification of skill entities and skill actions. Figure 6 displays how they are connected to the user, task and project classes of the OSH Project ontology. Skill entities serve as classes into which instances can be classified, e. g. an instance *3dprinter* of the *Skill Entity* subclass *Machine*. Skill actions are implemented as object property (relation) in the ontology. In Table 3 the relations of skill entities and skill actions are further sectioned.

Table 3: possible skill\_actions relating a user to a Skill\_Entity

Possible skill_action	Skill_Entity
operating, installing, maintaining, teaching, cost_estimating, designing	Machine
operating, installing, maintaining, teaching, designing	Tool
teaching, legal_information	Topic
technical_writing, cost_estimating	Material
teaching, cost_estimating, technical_writing, product_pricing	Process
teaching, cost_estimating, legal_information	Event

This structural division into skill action and entity differs from the ESCO hierarchy, in which these are usually combined (e.g. *maintainingMachine*, *operatingMachine*, ...). Such a division is on factual level more precise, since a skill usually involves an artefact and also an activity or action performed on/with it, like “teaching a topic”. On semantical level this division results in a stronger expressivity since now divided, both concepts (the skill\_action and the Skill\_Entity) can be semantically further described, enriched and also embedded into the semantic network with additional relations. In addition to such a division there are further advantages. On the one hand, there is no duplication of instances. As soon as a skill action changes on a skill entity, no new instance has to be created, but only a relation to the instance has to be changed. This means that only one semantically strongly enriched instance described by many relations is created instead of many semantically weakly enriched instances. On the other hand, skill entities can also be enriched with skill-independent attributes and relations, as is done in the case of project and issue tagging. If, for example, machines or tools are to be further specified, these do not have to be additionally created as concepts in the ontology.

Furthermore, there is the additional advantage that skill entities can be assigned to users even without a concrete skill action, such as *operating* or *maintaining*. On the one hand, this is an advantage, as information is often only available in fragments, so that an assignment is still possible; the generic parent-relation *skill\_action* is simply used. On the other hand, this also means that as soon as no results can be found for a query of a certain *skill\_action - Skill\_Entity* combination, it can simply be extended to the focus on the *Skill\_Entity*. For instance, someone is needed who understands the function of a

<sup>5</sup> <https://ec.europa.eu/esco/portal/home>

3D printer in order to build it (*building 3dprinter*). If no one can be found for this, it is possible to extend the query to someone who has, for example, already repaired or maintained a 3D printer (*repairing 3dprinter* or *maintaining 3dprinter* or even *skill\_action 3dprinter*). The knowledge of certain individual parts could also be helpful in building the printer.

### 3.2. Design notes

This section provides a deeper insight into the constitution of the code and how the methods are handled to instantiate and query the ontology.

#### 3.2.1. Class diagram

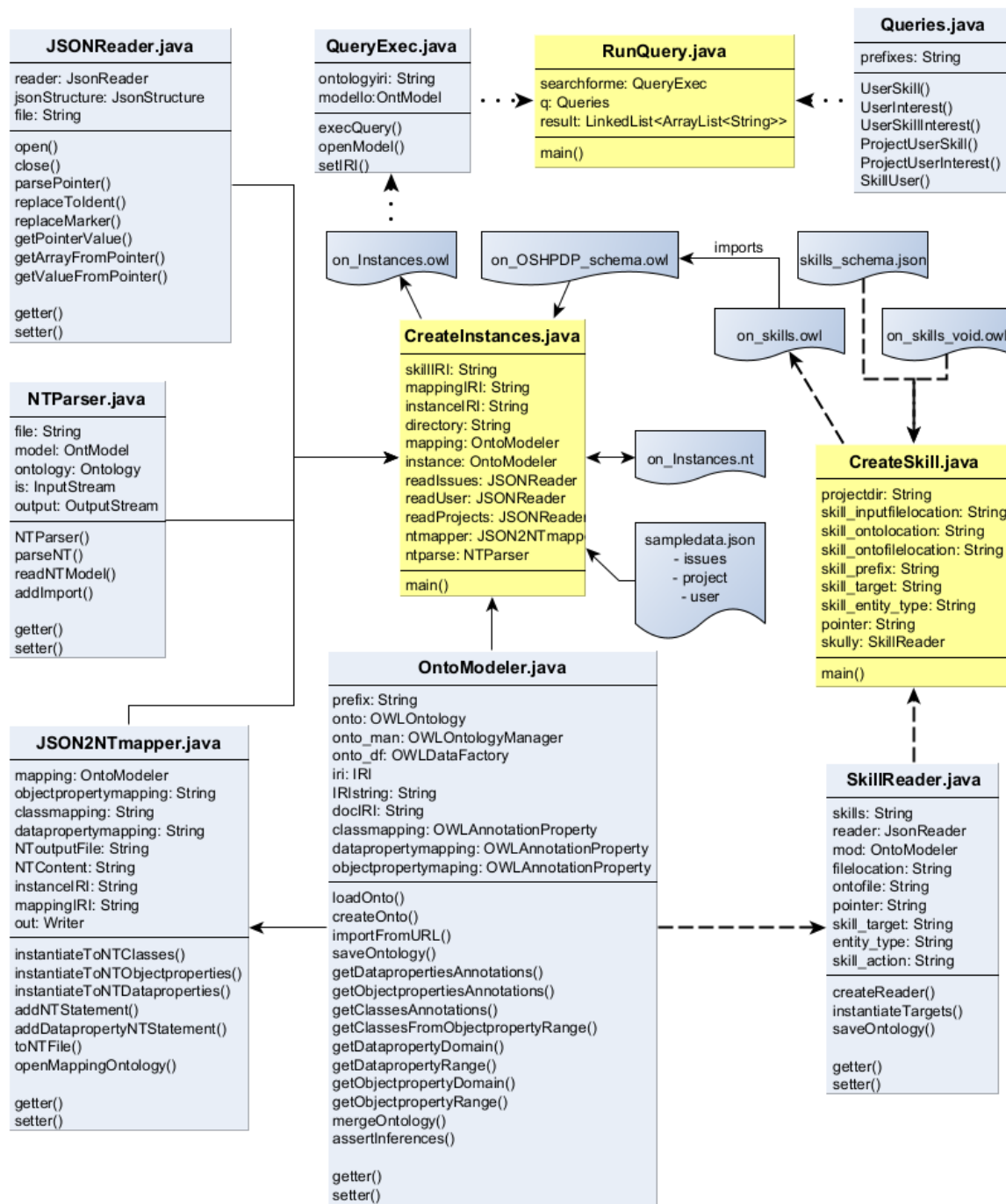


Figure 9: Demonstrator class structure

### *OntoModeler.java*

The *OntoModeler.java* class is used to create, load and save, handle and serialize OWL ontologies and ontology files. The main functionality in the demonstrator is, besides loading and saving the ontologies, to read out the mappings of the OSH project ontology, get information about the relating concepts (e. g. get the domain class of an object property) and to instantiate information from the JSON input files (this happens mainly in case of skill data instantiation). In addition, it is used for reasoning during the instantiation of the project data and saving the instances ontology.

### *SkillReader.java*

This class provides functionality to read skill information from the provided JSON input file (*skills\_schema.json*). It uses a JSON-pointer to specify the location of desired data in the JSON file and creates a JSON array. Defined key attributes are used to read out values from this array. Afterwards, the predefined skill schema (*on\_skills\_void.owl*) is loaded with the help of the *OntoModeler.java* class and the resulting values are instantiated as OWL ontology (*on\_skills.owl*) and saved.

### *CreateSkill.java*

This class holds the *main()* method for the skill instantiation process and coordinates it by passing parameters and calling functions of the *SkillReader.java* and *OntoModeler.java* classes. The parameters required for skill instantiation are initiated here in the beginning.

### *JSONReader.java*

The *JSONReader.java* class holds functionality to extract data from the JSON formatted inputs over JSON pointers. If pointers contain array (indicated by a “~” symbol as shown in section 3.3 Custom instantiation of the main report), there is functionality to parse those arrays into its entries. In addition, it also coordinates the connection of value information to its respective entry in the array. This ensures that an user instance created from the first (second, ..., n-th) entry is assigned to the user information found in the first (second, ..., n-th) entry of the array.

### *JSON2NTmapper.java*

This class uses the results of JSON data extraction from the combination of *JSONReader.java* and *OntoModeler.java* methods and converts them into a NT-file. N-Triples<sup>6</sup> (NT) is a plain text format for RDF graphs and serves for the instantiation as intermediate format from JSON to OWL translation.

### *NTParser.java*

This class loads an NT-file (in the instantiation case provided by methods from *JSON2NTmapper.java* class) and provides a method to enrich the file, e.g. with a statement to import the vocabulary of another ontology or setting different prefixes to the ontology. In addition, there are functions to convert the NT-file to another RDF based format.

### *CreateInstances.java*

This class holds the *main()* method for the project data instantiation. It provides all input files and locations. Connecting the *OntoModeler.java*, *JSONReader.java*, *JSON2NTmapper.java* and

---

<sup>6</sup> <https://www.w3.org/TR/n-triples/>

*NTParser.java* classes, it coordinates the calling of methods and handover of parameters and results of methods.

#### *Queries.java*

This class provides methods to return SPARQL query strings based on the described user flows. Some SPARQL query examples are given in Table 2 of the main document in section 3.2

#### *QueryExec.java*

The *QueryExec.java* class connects to an ontology and executes queries on it. In case of the demonstrators, the instantiated project data (*on\_Instances.owl*) is queried with the query strings provided by the *Queries.java* class.

#### *RunQuery.java*

This class holds the *main()* method for the querying process and uses SPARQL queries from the *Queries.java* class passing it to the *QueryExec.java* class.

### 3.2.2. Flow instantiation

The following diagrams visually facilitate the function of the main processes instantiation and querying. The instantiation process works differently for the skill instantiation and the project data instantiation. Both are shown below, for skill instantiation in Figure 10 and for the instantiation of the project data in Figure 11 and Figure 12. The flow charts are provided in a sequential structure, indicated by the process arrows. At the same time the different classes used are displayed in a swim lane fashion, indicated by lines without arrows. Before starting the instantiation process, it is possible to change needed variables, shown in the manual input section of the flow charts. For the user flows of the project, all variables are set and the instantiation methods were executed resulting in a fully instantiated ontology, which is ready to be queried.

The sequence for the instantiation process of the ontology in Figure 11 and Figure 12 is shortened to a level of general understanding. To get a more elaborate insight of the explicit code function, the project repository holds a complete flow of all functions used, necessary to understand it. However, a detailed presentation of self-explanatory methods has been omitted (e.g. setter methods). Additionally, the methods are mostly described in the commented code as well.

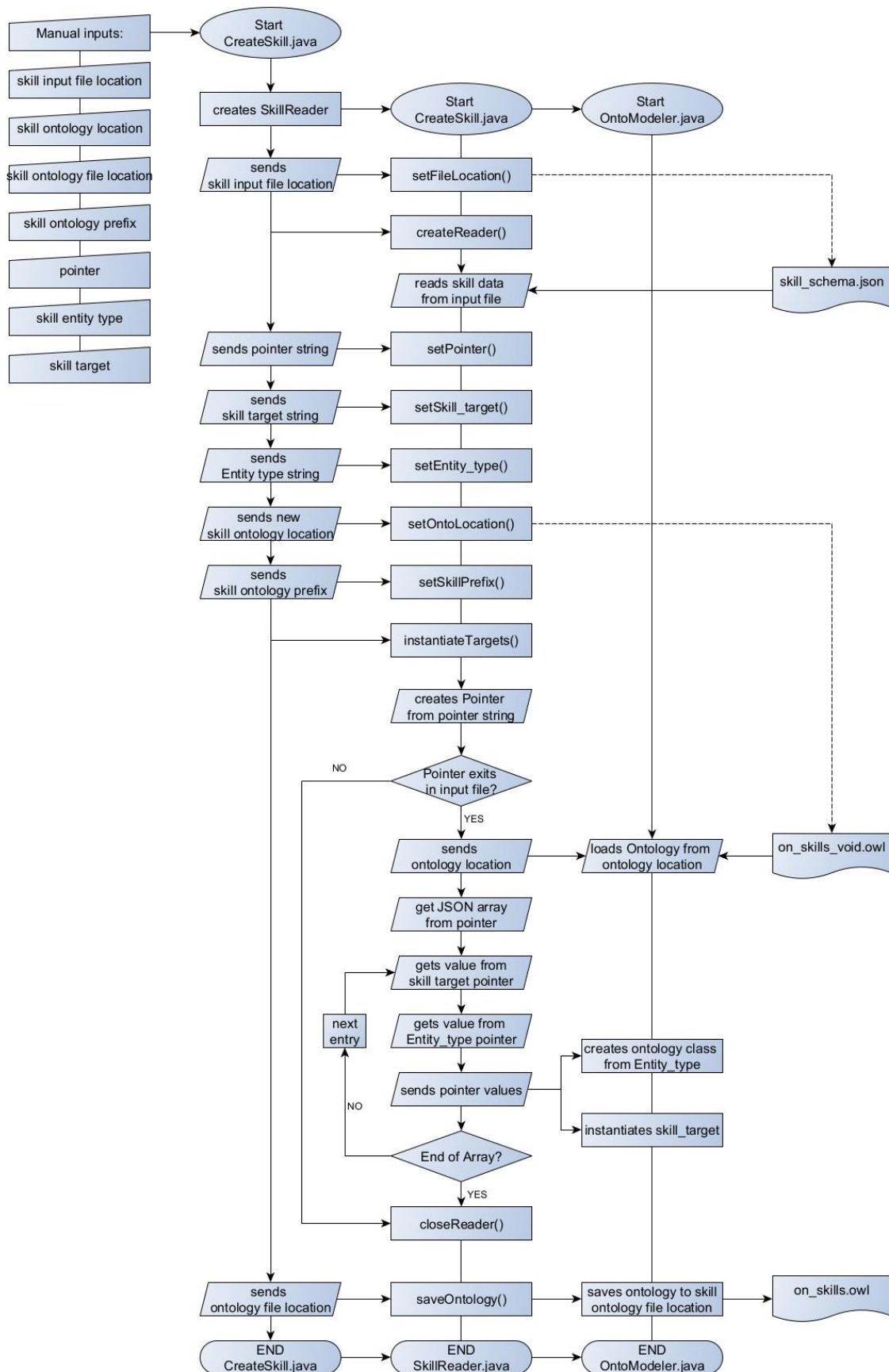


Figure 10: Skill instantiation flow

The skill instantiation is additionally presented shortly in writing. The CreateSkill.java class holds the main method and is responsible for the process flow.

At the beginning an instance of the SkillReader.java class is created and the location for the JSON input file is given. This input file is read by a reader that now holds all information needed. Now the different pointers are set:

- The *pointer* variable indicates which section of the input file is necessary for the instantiation
- The *skill target* variable indicates which values are instantiated as individuals (that counts e. g. for *3d-printing*)
- The *skill entity type* variable shows how to classify a relating *skill target* variable (e. g. *3d-printing* is an individual of the skill entity type class *Process*)

After setting all variables, the instantiation process begins (instantiateTargets()):

- 1) A pointer is created from the *pointer* string variable
- If the pointer exists, the void skill ontology is loaded to be instantiated.
  - The data array from the pointer variable is read
  - For each entry in the array a skill target and skill entity type are read and instantiated. A skill target variable is instantiated as individual of the class indicated by the relating skill entity type.
  - After instantiation of all skill targets, the instantiated ontology is saved.



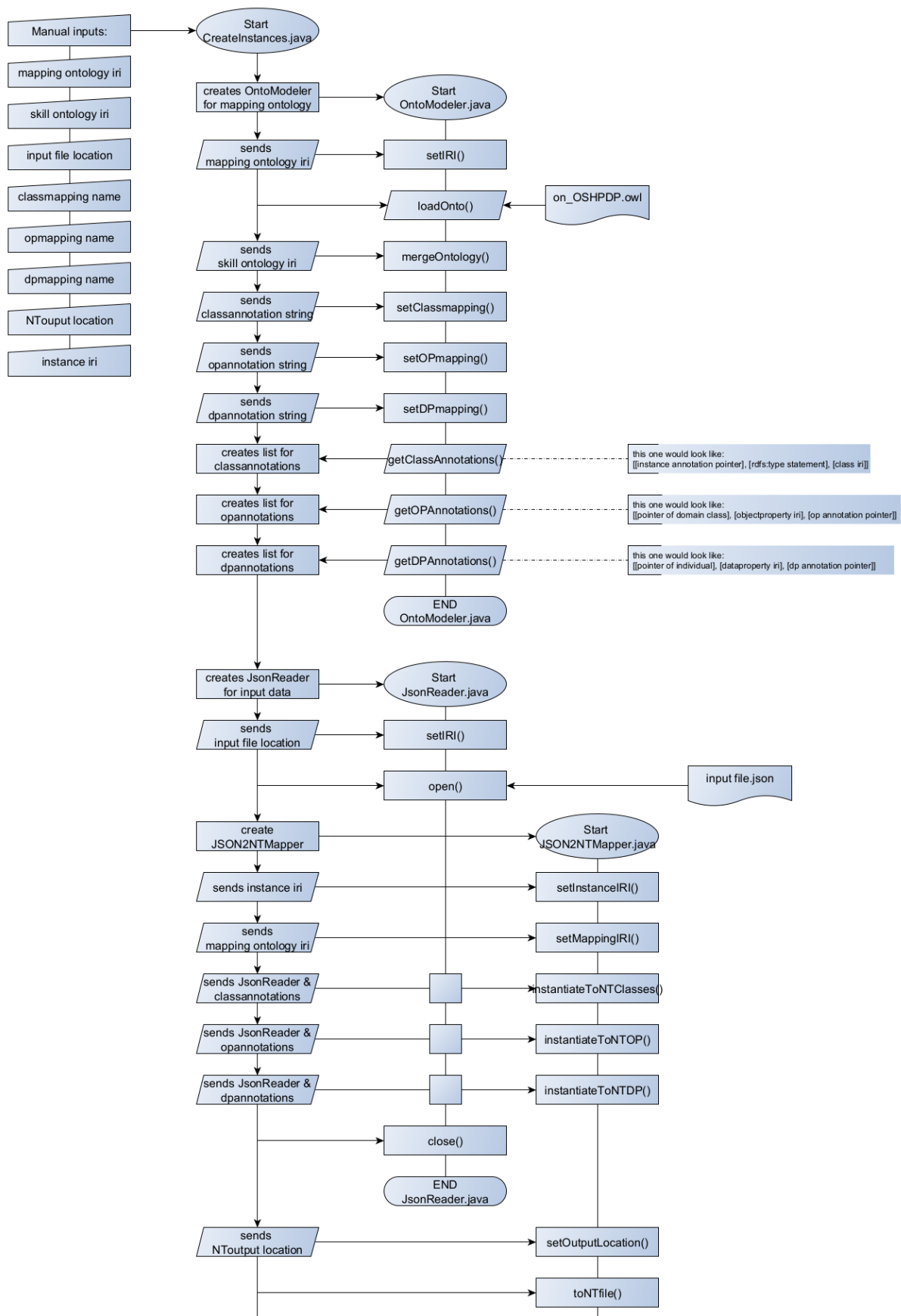


Figure 11: Flowchart Instantiation (1/2)

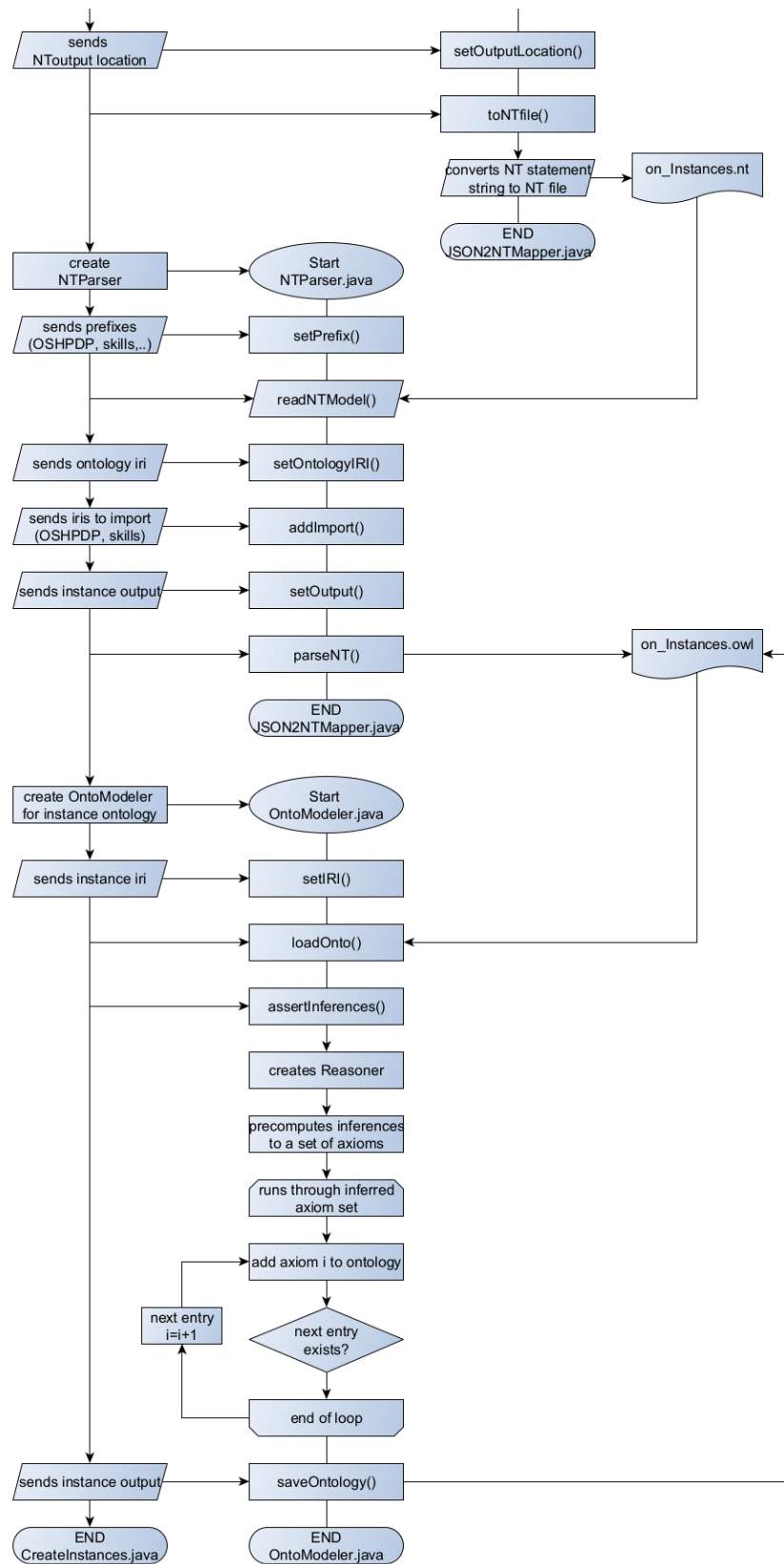


Figure 12: Flowchart Instantiation (2/2)

The instantiation process is briefly described in four main parts. The repository contains a much more detailed representation of some functions in the flow<sup>7</sup>, which is not beneficial for the understanding here and would exceed the scope of the explanations of this work.

#### 1) Getting the mapping annotation properties from the OSH project ontology

This part relates in the flow chart to the section between the steps “*Start CreateInstance.java*” and “*END OntoModeler.java*”. After setting the manual inputs, the instantiation process starts with the initialization of an *OntoModeler.java* object in the *CreateInstances.java* class. After that the following short steps take place:

- The IRI of the mapping ontology is set and the ontology file is loaded
- The mapping ontology is merged with the ontology found behind the *skill ontology IRI* (here the skill ontology)
- The mapping annotations are set, which should be searched in the following process. (*setClassmapping()*, *setOPmapping()* and *setDPmapping()*)

After setting what mapping annotations should be searched, the mapping annotations of every class, object property and data property are read out of the ontology and the results are saved into lists. Each list contains three arrays and is arranged as follows for each of the subsequent cases:

- Class mapping annotations: The first array contains all mapping annotation pointers for the individuals to be instantiated. The second array simply contains an “*rdf:type*” statement and the third array holds the respective class IRI of the instances class.
- Object property annotations: The first array contains the pointers of the domain class from the respective object properties, which’s IRIs are stored in the second array. The third array stores the object property annotation pointers of the range classes from the respective object properties.
- Data property annotations: The first array contains the pointers for the individuals to be instantiated. The second array contains the data property IRIs and the third array contains the respective data property annotation pointers.

#### 2) Creating an NT file from the mapping annotation pointers

The explanations in this part refer to the steps from “*Start JSONReader.java*” to “*END JSON2NTMapper.java*”. After the initialization of a *JSONReader.java* object, the file location of the JSON inputs is set and the file is opened. A *JSON2NTMapper.java* object is created and the mapping ontology IRI, to load the existing OSH project ontology, and the instance IRI, in which the new instances are to be stored, are set. With the *JSONReader.java* object annotation pointers from the lists of class-

---

<sup>7</sup>[https://github.com/OPEN-NEXT/WP3\\_Skillmatching/raw/main/files/Flowchart\\_create\\_instances\\_complete.png](https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/files/Flowchart_create_instances_complete.png)

, object property- and data property annotations created in the first step are replaced with their corresponding values. With the values and the IRIs from the ontologies, NT statements are created and saved as NT file.

### 3) Converting the NT file into an RDF format

This section concentrates on steps between “*Start NTParser.java*” and “*END NTParser.java*”. When created, the *NTParser.java* creates an ontology model, which will be enhanced with the further described information. Necessary prefixes are added to the ontology model with the *setPrefix()* method. The *readNTModel()* method reads the NT file with instances information from its location that was handed over to the constructor at creation. After setting the ontology IRI, adding necessary statements to import concepts and vocabulary from other ontologies and setting the output file location, the ontology model is saved in OWL format.

### 4) Reasoning over the ontology and assert inferences

The last section focuses on the steps from “*Start OntoModeler.java*” to “*END OntoModeler.java*”. After initializing an *OntoModeler.java* object, the instance ontology is loaded from its IRI. The *assertInferences()* method creates a reasoner, that precomputes inferences, creating new axioms for the ontology. A loop runs through the set of new axiom, adds it to the ontology and saves it.

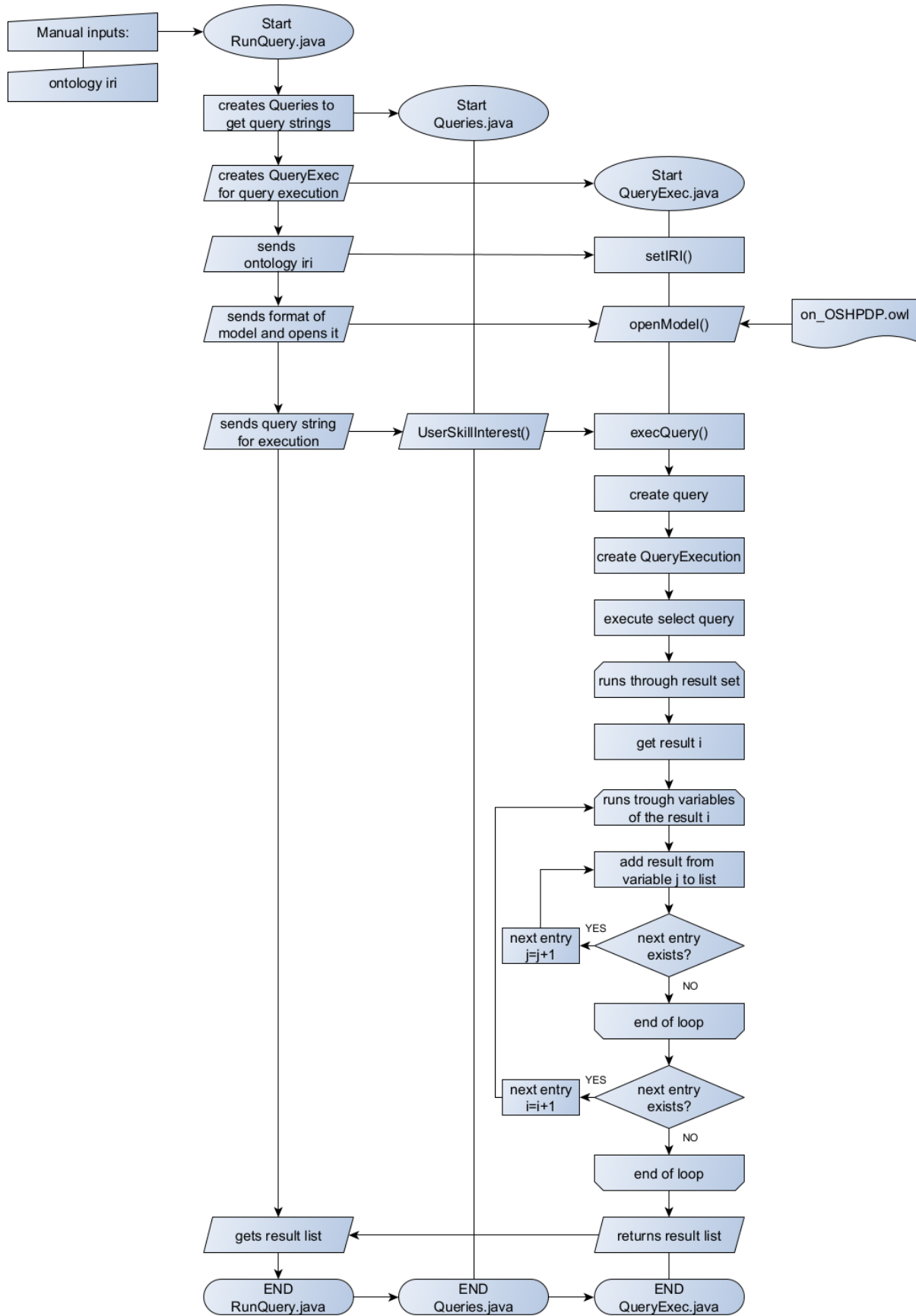


Figure 13: Query execution flow

The query execution flow is shown in Figure 13 and starts with the *RunQueries.java*. In this class, the ontology IRI to be loaded and queried is provided as string variable. After initiation of a *Queries.java* and a *QueryExec.java* object, the ontology IRI is set for the query execution and the ontology model is loaded. The *Queries.java* class provides methods to return query strings for the execution on the ontology. In Figure 13 the example of the *UserSkillInterest()* query generation method is shown in the process. This can be replaced with other query generation methods from the *Queries.java* class. The generated query string is handed over to the query execution method of the *QueryExec.java* class. To execute the query a *Query* and a *QueryExecution* type variable are created and the result set from the query is generated. The result set is run through and the results for every variable is saved into a result list. The number of variables depends on the number of variables in the SELECT clause of the query. At the end the result list is post-processed to match a table like layout.

### 3.3. Installation

There is no specific installation needed. Depending on the use, the repository may have to be cloned. That is the case, when the mapping annotation properties in the ontology needs to be changed or if the query functionality provided in the demonstrator is used. For this general knowledge about Git is required. Furthermore, the structure of RDF and SPARQL should be known for the query use of the ontology. Interested developers, that need to change, adapt or improve the ontology, should be familiar with OWL and its extensions in contrast to RDF if aspiring reasoning or axiomizing. Table 4 and

Table 5 document the used software stack and the dependencies in software code.

Table 4: Used software stack

Software or Tool	Purpose
GitHub	Ontology hosting and demonstrator documentation
Protégé Editor <sup>8</sup> v5.5.0	Ontology modelling
GraphQL API	WIF data extraction
Eclipse IDE (v 2020-03 R.0) with JRE1.8	Coding and development of the demonstrator, including ontology files, JSON inputs and connection to GitHub repository

Table 5: Software code dependencies

Dependency package	Purpose
ONTAPI <sup>9</sup> v2.1.0	JAVA package for ontology handling. It combines the JAVA ontology APIs OWL API and JENA API <sup>10</sup> . This was used for parsing and serializing NT data during the instantiation process and for the SPARQL query.
OWL API <sup>11</sup> v5.1.14	Java package for handling OWL ontologies. This was used to extract the mapping annotation properties and to create the skill instances in the on_skills.owl

<sup>8</sup> <https://protege.stanford.edu/>

<sup>9</sup> <https://github.com/owlcs/ont-api>

<sup>10</sup> <https://jena.apache.org/>

<sup>11</sup> <https://github.com/owlcs/owlapi>

Hermit v1.3.8.510	JAVA OWL Reasoner to assert inferences
javax.json v1.1.2	JAVA package for JSON processing, especially providing the use of JSON pointers.

For changes in the mapping process, handling of the JAVA ontology APIs, OWL API and JENA API are used. They are JAVA implementations to build and handle semantic applications and provide functions to create, manipulate and serialize ontologies. To understand the functioning of the demonstrator, a short example is detailed below. It is oriented on the software stack shown in Table 4. Changing and further handling of the demonstrator is explained in the step-by-step examples section of the main document. For further developmental changes, refer to the flowcharts in the previous design notes section for an in-depth understanding.

The following steps can be followed to run the demonstrator:

- 1) Visit the GitHub repository: [https://github.com/OPEN-NEXT/WP3\\_Skillmatching](https://github.com/OPEN-NEXT/WP3_Skillmatching)
- 2) Copy the repository link

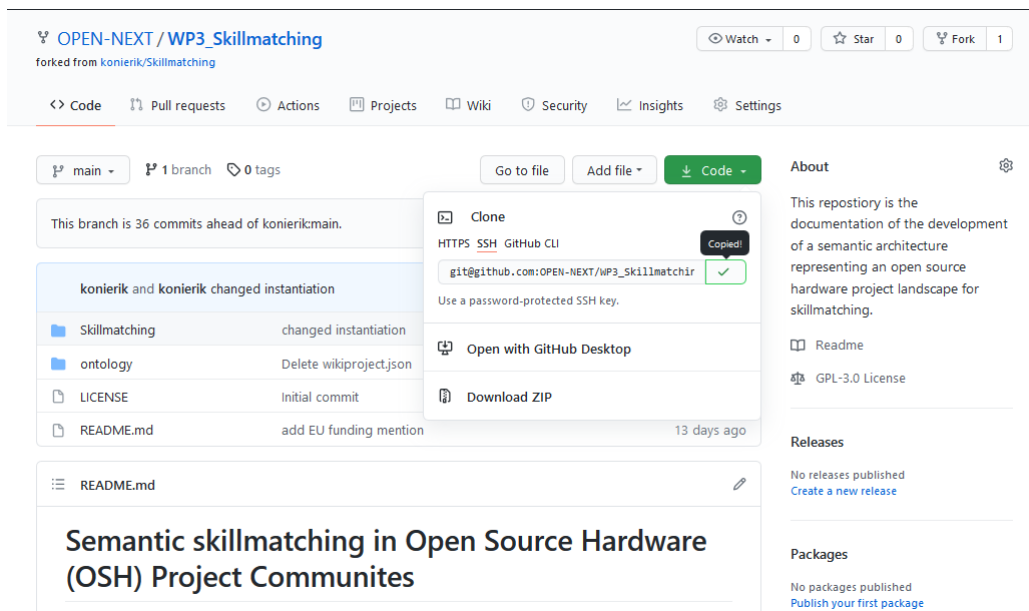


Figure 14: Copying the link of the GitHub repository page

- 3) If not already done, an SSH key should be created to connect to the repository. Please refer to the *Connecting to GitHub with SSH*<sup>12</sup> guide
- 4) Open the Eclipse IDE and choose a workspace

<sup>12</sup> <https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh>

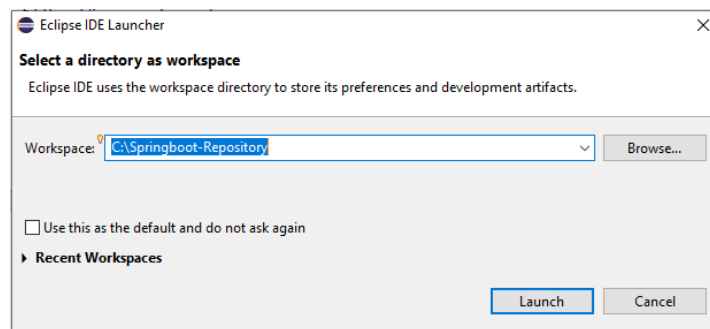


Figure 15: Starting Eclipse and choosing a workspace

5) Check if Eclipse uses the SSH key from GitHub

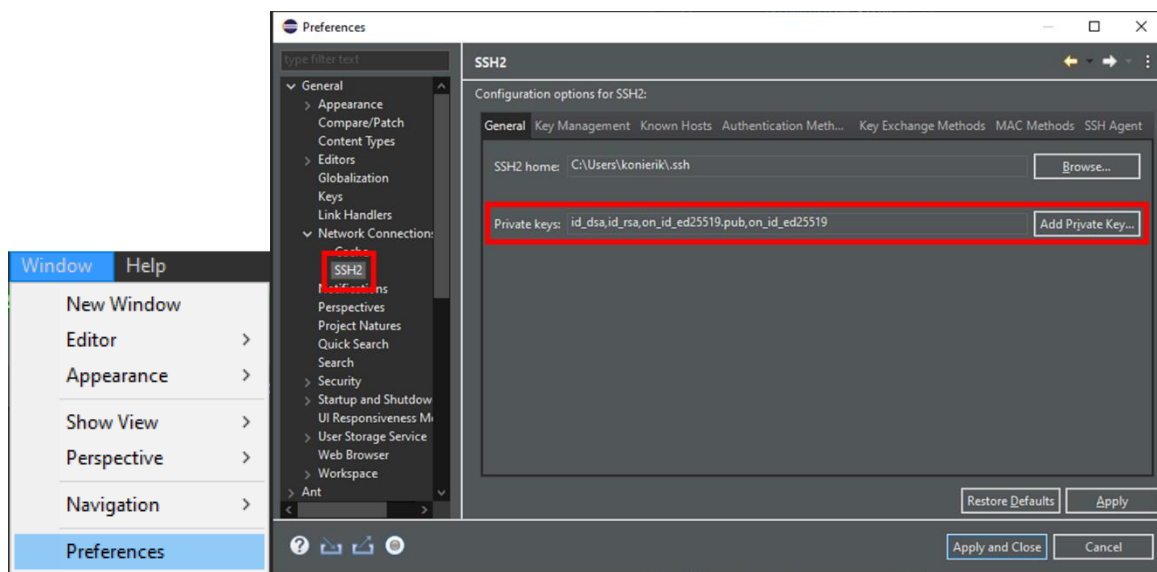


Figure 16: Setting the SSH2 key in the preferences

The SSH keys used in Eclipse can be found under *Window>Preferences>General>Network Connection>SSH2*.

If the key is not found there, it has to be added from the location it was saved during step 5) of this section.

6) Open the git perspective

Following *Windows>Perspective>Open Perspective>Other...*



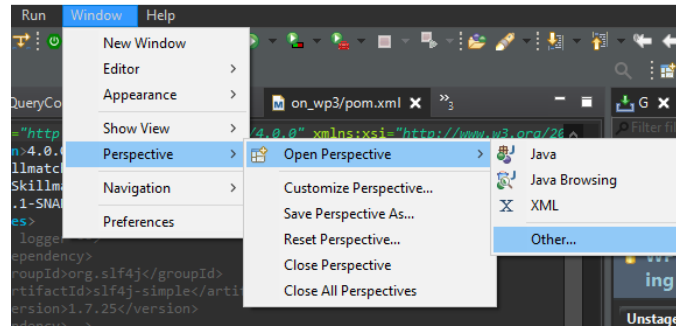


Figure 17: Open Git perspective in Eclipse (1/2)

Select *Git*.

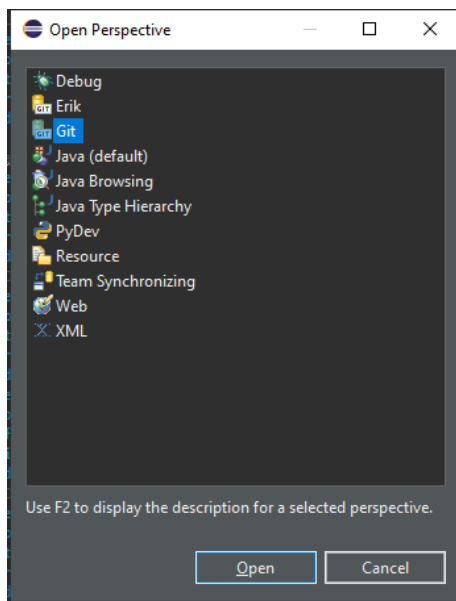


Figure 18: Open Git perspective in Eclipse (2/2)

7) Click on *Clone a git repository* in the Git Repositories tab.

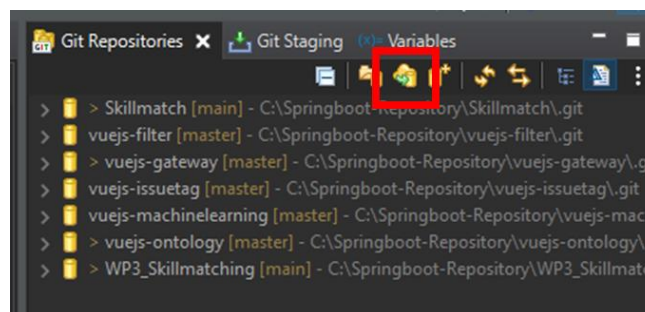


Figure 19: Clone a Git repository from the Git repositories tab

Afterwards the dialog box for cloning a repository then opens.

8) Specify the repository to clone

After inserting the repository link into the URI field, the other fields should be filled automatically. If this does not happen, the fields have to be filled according to Figure 20.

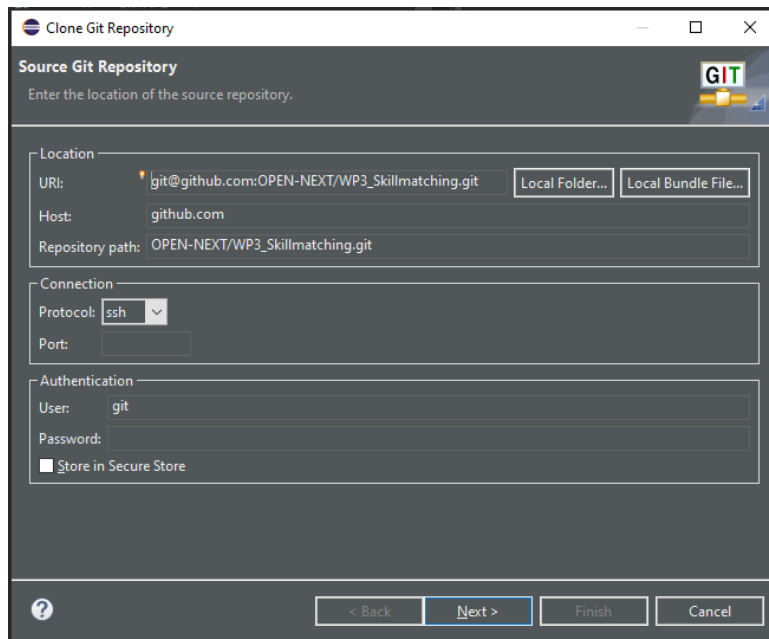


Figure 20: Inserting the repository URI to clone

Click *Next*.

- 9) Select the branch. The usual case is to clone the main branch.

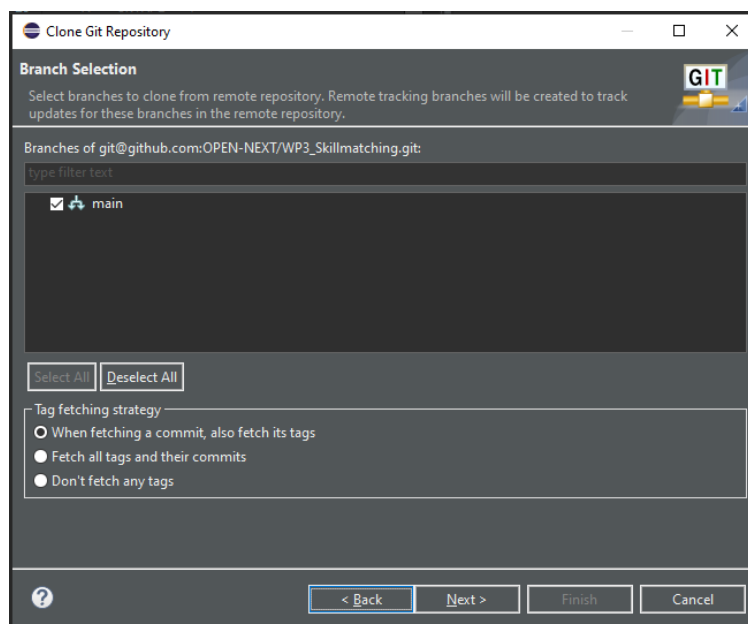


Figure 21: Selection the branch to clone

Click on *Next*.

- 10) Choose the local destination of the Git repository by filling the *Directory* field.

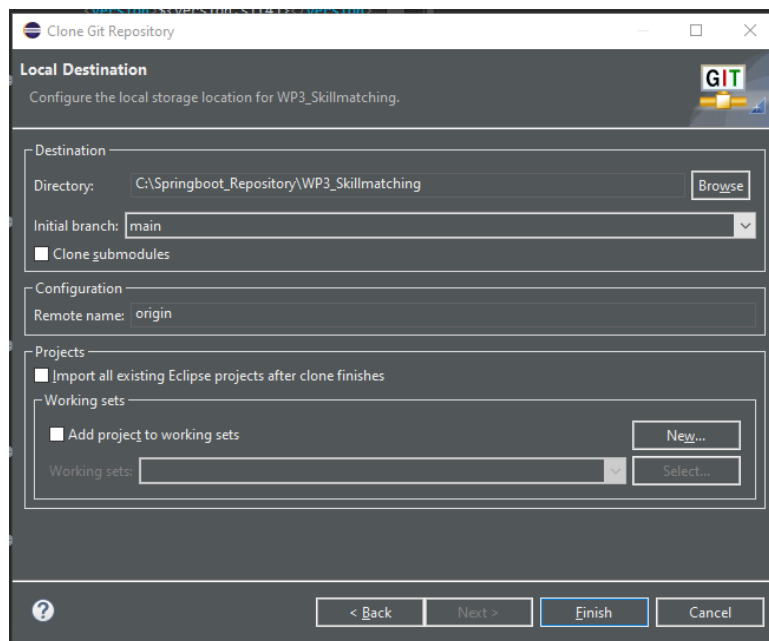


Figure 22: Choosing a local directory for the git repository

Click on *Finish*. The repository *WP3\_Skillmatching* should now be displayed in the Git Repositories tab (cf. Figure 19).

Now a Maven project needs to be imported that integrates the Git.

### 11) Import a Maven project

Following *File>Import>Existing Maven Projects*

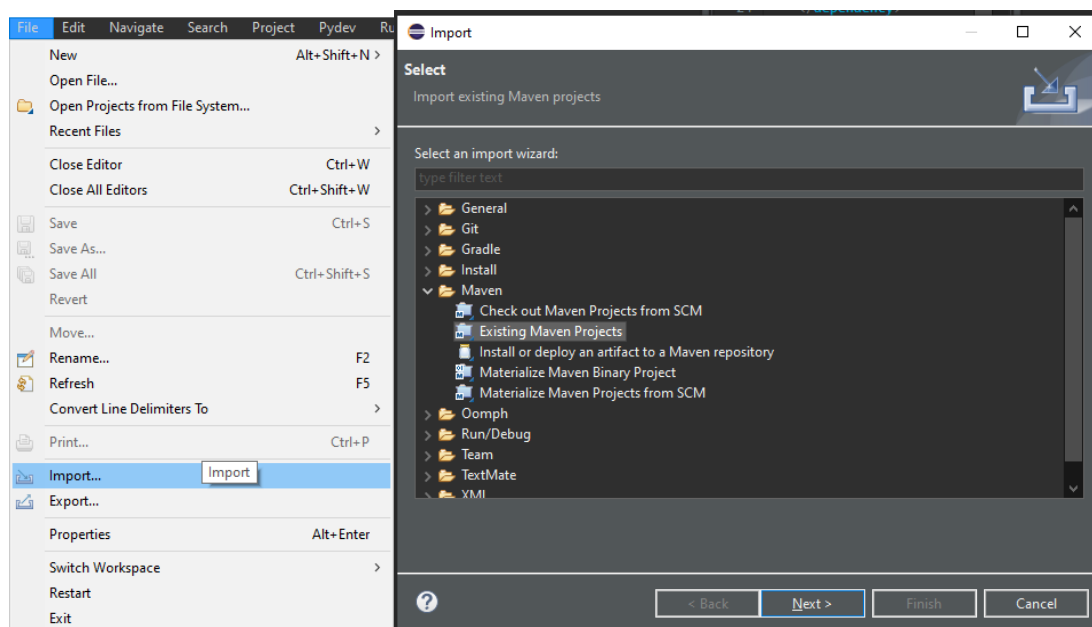


Figure 23: Importing existing Maven project

Choose the root directory with *WP3\_Skillmatching*. For this manual another project *WP3\_Skillmatching\_manual* was created to be imported which is why it is displayed in the figures. For the use of the skill matching demonstrator the *WP3\_Skillmatching* but has to be used.

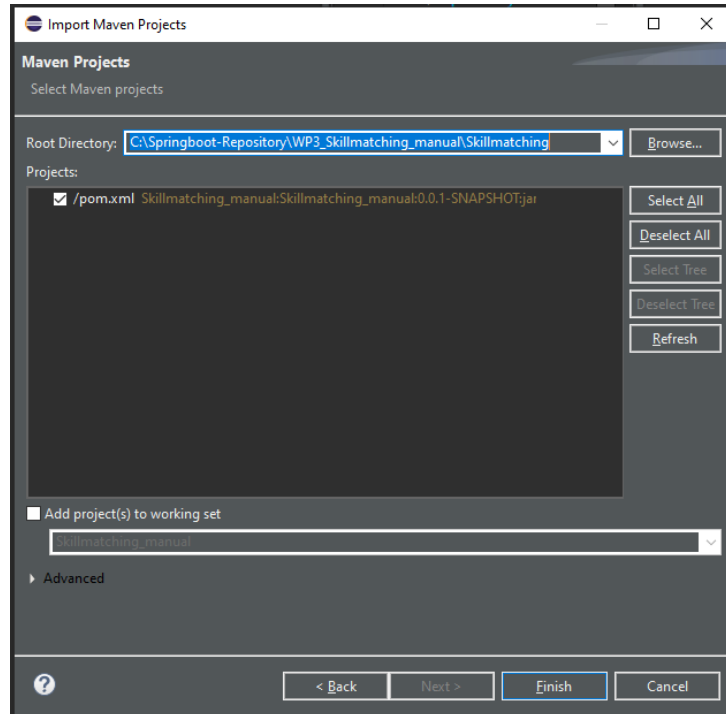


Figure 24: Choosing root directory

Click on *Finish*.

The Maven project will appear in the *Project Explorer* tab.

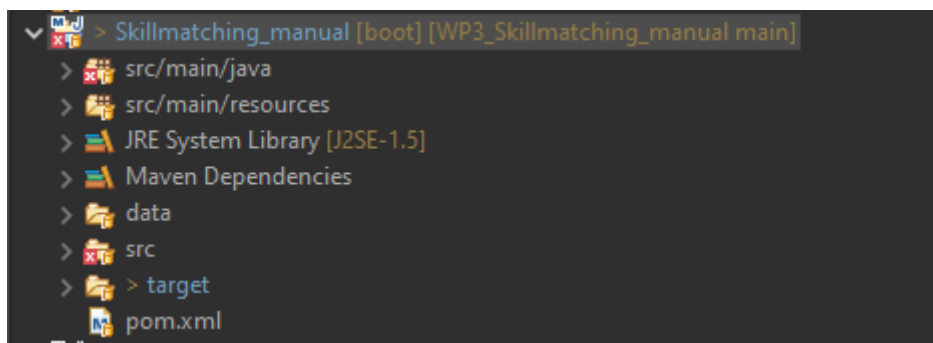


Figure 25: Imported project in the project explorer tab

If there are some errors right away, most likely the JAVA compiler version has to be set to 1.8.

## 12) Set JAVA Compiler Version

This can be done in the properties of the project. (Right click on the project)

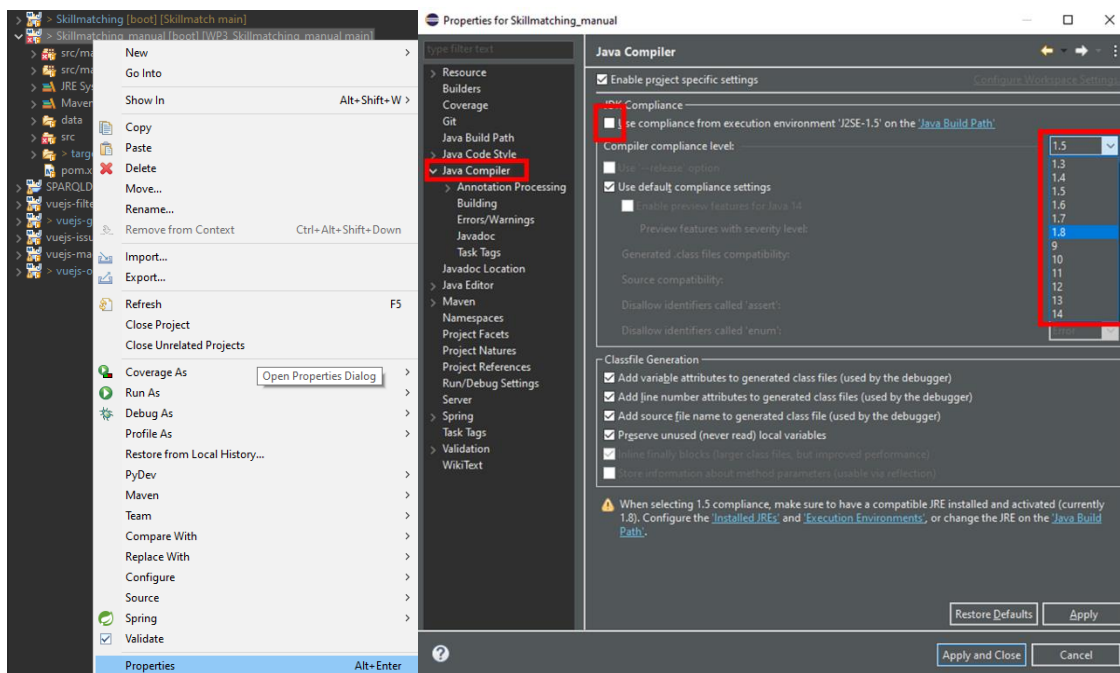


Figure 26: Setting JAVA Compiler level

In the *Java Compiler* tab the *Use compliance from execution environment 'J2SE-X.X' on the Java Build Path* needs to be unchecked and the *Compiler compliance level* has to be set to 1.8.

Click *Apply and Close*.

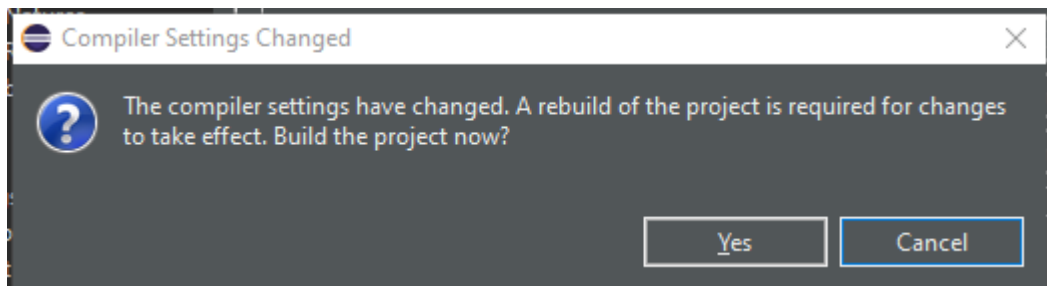


Figure 27: Agreeing to the project build

The rebuild of the project has to be accepted by clicking on *Yes*.

If an error on the JUnit import statement occurs in the *OntoModeler.java* class, the library has to be added to the project.

```
import static org.junit.Assert.assertNotNull;
```

Figure 28: JUnit import error

The library can be added in the Java Build Path by right clicking the project and following *Properties>Java Build Path>Libraries>Add Library>JUnit>JUnit5*.

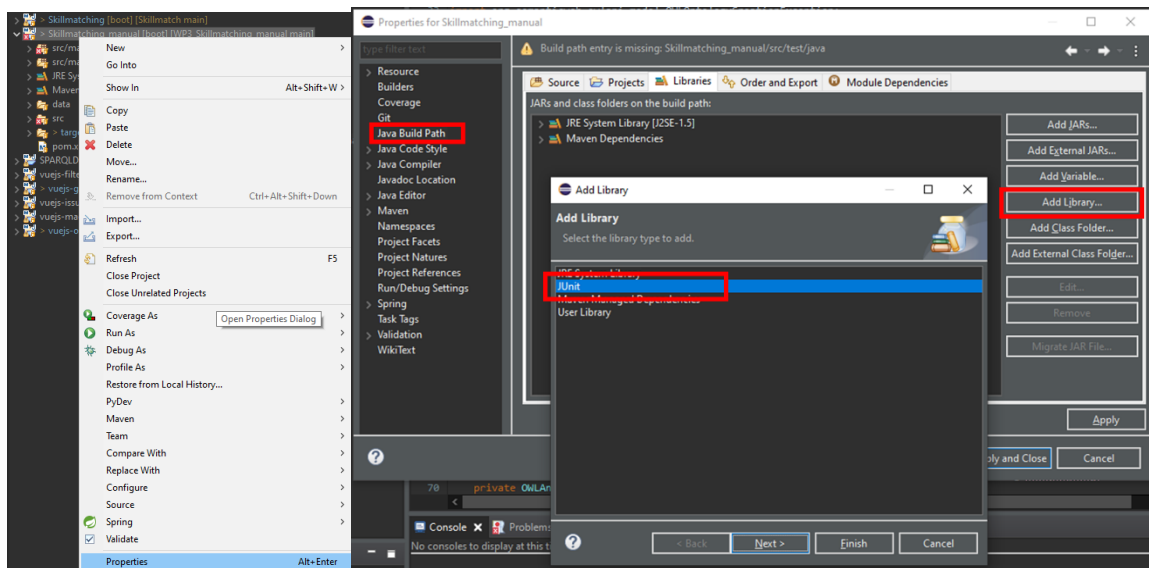


Figure 29: Adding JUnit library

Click *Next*. Choose *JUnit5*. Click *Apply and Close*.  
 Now the demonstrator is ready to use.

### 3.4. Ontologies

In this section, basics of ontology modelling with respect to frameworks, languages and documents used are detailed.

#### 3.4.1. Resource Description Framework and Web Ontology Language

The Resource Description Framework (RDF) is used to model knowledge on the web. It uses triple patterns in “subject-predicate-object” style to connect subject and object resources (uniquely linked via uniformed resource identifier (URI) via relations, so called properties. The RDF schema (RDFS) builds upon RDF and provides schematic definitions about predefinitions. Those are e.g. the definition of constants like class, properties, range (called the object resource of a property) or domain (called the subject resource of a property). Those triple patterns create a net of knots and relations in a manner of graph theory. The web ontology language (OWL), the most used language to model semantics, builds upon RDF and extends it with more expressions making it semantically richer and allows reasoning capabilities (Pan et al. 2017).

(Antoniou and van Harmelen 2009) provides a comprehensive overview about differences between RDF and OWL. Knowledge that is necessary for a basic understanding is the structure in OWL ontology, which is explained and illustrated below:

Classes are arranged in a hierarchy and connected to each other via object properties. These can also be arranged in a hierarchy. For each property, the domain class (the subject of the relation) and the range class (the object of the relation) are defined. Instances are realizations of resources/individuals within an ontology and are classified. They can be connected to other instances by using object properties. Another type of property, a Data property links an instance of a class to a value of a primitive data type. Data properties are used to describe instances more precisely. The domain of a data property is the class to which the instance belongs to. The range, as described, is a primitive data type. These relationships are shown in Figure 30: The instances “User1” of class User and “ProjectX” of class Project are connected to each other via “member\_of” and “has\_member” object properties.

Both instances are further described with the data properties “Name” and “ID”. The dotted lines in Figure 30 show the relation between instantiated concepts and the class, and property structure of the ontology.

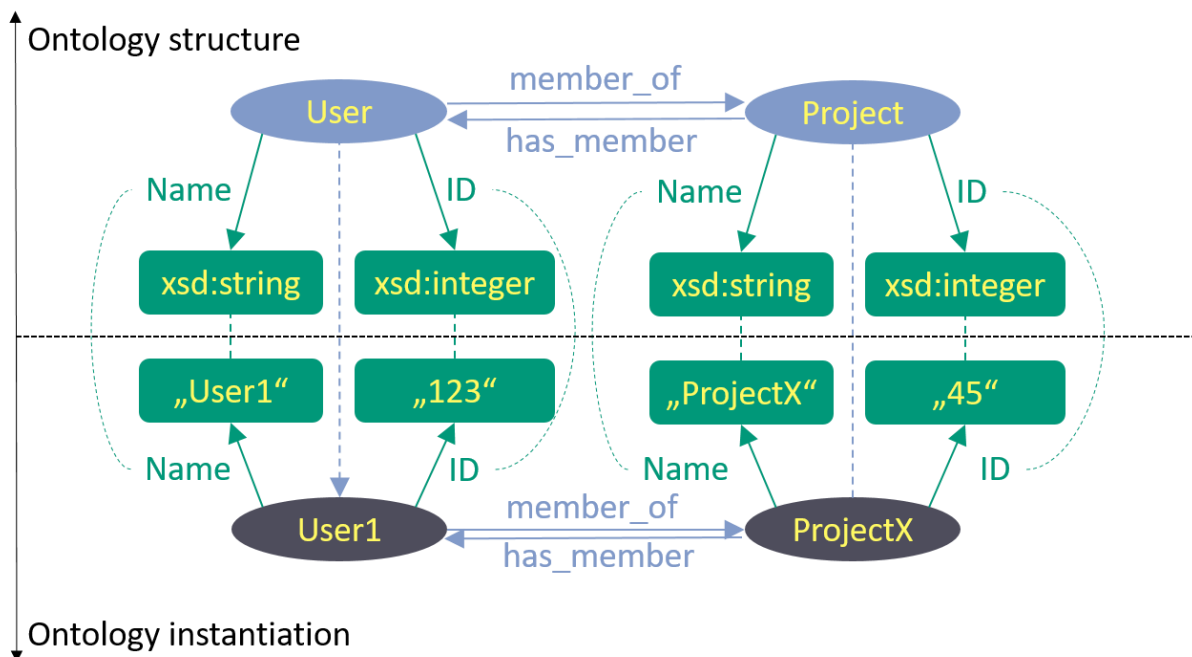


Figure 30: Instantiation example

### 3.4.2. OWL documents

The constitution of an OWL file is simple. Every OWL file begins with a header that specifies the ontologies terms, with the following components:

- The ontology namespace and version is defined. The namespace of an ontology is used to uniquely identify it. This is advantageous because of disambiguation when using several vocabularies in one ontology. Since OWL is based on RDF, there are some RDF/S namespaces defined in the ontology header right away, so standard RDF vocabulary can be used without import (more about the import will follow in the next points).
- Prefixes used in the Ontology are displayed. Generally, a prefix refers to an ontology namespace. Used in the ontology, the prefix is exchanged and so refers to the location of the used resource. An example is given for the rdfs:type property. The prefix “rdfs” refers to its namespace and the location “<http://www.w3.org/2000/01/rdf-schema#>”. Using the rdfs:type property in the ontology, the prefix is resolved and the property <http://www.w3.org/2000/01/rdf-schema#type> is used in the statement.
- An (optional) import statement is added. The import statement refers to another ontology namespace and causes the usability of the vocabulary from the imported ontology. All ontology vocabularies used, need to be imported at first. (Exceptions are the standardized RDF vocabularies mentioned above.)

The header of the developed ontology is shown in Figure 31.

```

<?xml version="1.0" ?>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
  xml:base="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPPD_schema.owl"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  ontologyIRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPPD_schema.owl"
  versionIRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPPD_schema.owl/0.1">
  <Prefix name="" IRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPPD_schema.owl#" />
  <Prefix name="do" IRI="http://purl.org/dc/elements/1.1/" />
  <Prefix name="okh" IRI="http://purl.org/oseg/ontologies/osh-metadata/0.1/base" />
  <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
  <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
  <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace/" />
  <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
  <Prefix name="foaf" IRI="http://xmlns.com/foaf/0.1/" />
  <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
  <Prefix name="oshpd" IRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPPD_schema.owl#" />
  <Prefix name="skills" IRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_skills.owl#" />
  <Prefix name="doterms" IRI="http://purl.org/dc/terms/" />
  <Prefix name="wikifactory" IRI="https://wikifactory.com/api/graphql/" />
  <Import>https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_skills.owl</Import>

```

Figure 31: Ontology header

After the header, possible annotations on the ontology follow, such as creator. However, these do not contribute to the function, so they are only briefly mentioned here for completeness.

Now following are the declaration statements for the classes, properties and individuals, that occur in the ontology, shown in Figure 32 and Figure 33 :

```

<Declaration>
  <Class IRI="#Project"/>
</Declaration>
<Declaration>
  <ObjectProperty IRI="#has_member"/>
</Declaration>
<Declaration>
  <DataProperty IRI="#Project_title"/>
</Declaration>

```

Figure 32: Declaration axioms for (a) Class Project (b) object property has\_member (c) data property Project\_title

```

<Declaration>
  <NamedIndividual IRI="#CAD_file"/>
</Declaration>
<Declaration>
  <AnnotationProperty IRI="#wif_issue_1_cmap"/>
</Declaration>

```

Figure 33: Declaration axiom for (a) individual CAD\_file (b) annotation property wif\_issue\_1\_cmap

After that, the declared concepts are put into relation and further defined with additional axioms. That are e.g. setting the domain and range of properties, adding subclass/subproperty axioms or cardinality constraints for properties. Some of them used in the ontology, that are necessary to understand the main report, are displayed in the following figures and shortly explained.

```

<SubClassOf>
  <Class IRI="#Author"/>
  <Class IRI="#User"/>
</SubClassOf>
<DisjointClasses>
  <Class IRI="#Project"/>
  <Class IRI="#User"/>
</DisjointClasses>

```

Figure 34: Axioms to set (a) subclass relation and (b) disjointness of classes

Figure 34 (a) hierarchizes the Author class as subclass of User. Figure 34 (b) sets the classes Project and User as disjoint, i.e. those classes do not share any individuals. (In other words, an instance User cannot be a Project at the same time)

```

<ClassAssertion>
  <Class IRI="#programming_languages"/>
  <NamedIndividual IRI="#C"/>
</ClassAssertion>

```

Figure 35: Classifying an individual



Figure 35 classifies the individual “C” as instance of the lass “programming\_languages”.

```
<InverseObjectProperties>
  <ObjectProperty IRI="#has_member"/>
  <ObjectProperty IRI="#member_of_project"/>
</InverseObjectProperties>
```

Figure 36: Defining inverse properties

Figure 36 defines the object property “member\_of\_project” as inverse of the “has\_member” property. Since object properties are defined as unidirectional, an inverse property needs to be defined. The domain of a property therefore is the range of the inverse property and vice versa.

```
<ObjectPropertyDomain>
  <ObjectProperty IRI="#has_member"/>
  <Class IRI="#Project"/>
</ObjectPropertyDomain>

<ObjectPropertyRange>
  <ObjectProperty IRI="#has_member"/>
  <Class IRI="#Member"/>
</ObjectPropertyRange>
```

Figure 37: Axioms defining the (a) domain and (b) range of a property

Figure 37 shows axioms setting (a) the Project class as domain of the object property “has\_member” and (b) the “Member” class as range. The domain and range definition for a data property works similar but the range refers to a value. (The “<Class IRI=../>” statement therefore would be changed to refer to a datatype <Datatype abbreviatedIRI="xsd:string"/>).

```
<AnnotationAssertion>
  <AnnotationProperty IRI="#wif_issue_1_cmap"/>
  <IRI>#Project</IRI>
  <Literal>/data/issues/result/edges/~node/project/id</Literal>
</AnnotationAssertion>
```

Figure 38: Defining the annotation property wif\_issue\_1\_cmap

Figure 38 shows how the annotation property “wif\_issue\_1\_cmap” is further defined. The property is related to the “Project” class and contains a literal value for the JSON pointer.

```
<SubClassOf>
  <Class IRI="#Creator"/>
  <ObjectUnionOf>
    <ObjectMinCardinality cardinality="1">
      <ObjectProperty IRI="#creator_of_project"/>
      <Class IRI="#Project"/>
    </ObjectMinCardinality>
    <ObjectMinCardinality cardinality="1">
      <ObjectProperty IRI="#creator_of_task"/>
      <Class IRI="#Task"/>
    </ObjectMinCardinality>
  </ObjectUnionOf>
</SubClassOf>
```

Figure 39: Property restriction on class Creator

Figure 39 gives an example on a more complex axiom referring the class “Creator”. The class is restricted with boolean combination “union” of two cardinality constraints, meaning one of them needs to be true. The cardinality constraints each state that an instance of “Creator” has to be at least

one property referring to an instance of “Project” (property “creator\_of\_project”) or “Task” (property “creator\_of\_task”). The whole property restriction states that to be a creator, one has to be the creator of at least one project or one task.

## 4. Collaborative Production – Identifying production metadata

### 4.1. Development method

To identify metadata relevant for the replication of OSH products, one-on-one feedback sessions were carried out with makers from the OPEN\_NEXT consortium and the OSH community. They were invited by open calls. Volunteers have been asked for a self-assessment of their level of expertise in manufacturing hardware; each voluntary gave a number with 1 being the lowest (beginner level) and 5 being the highest (expert).

For the input section, makers received a short introduction into the concept of the Wikibase instance and were confronted with two questions:

1. What the production categories of interest (e.g. 3D-printed) when dealing with OSH are?
2. What are the corresponding data fields (e.g. material, size) in those categories that you would need to assess whether or not it makes sense for you to produce the corresponding part/module yourself or e.g. order it somewhere else?

Every maker was granted editing rights (“developer access”) to a Markdown document on GitHub. They were asked to write their input directly in that document by following the prepared structure. To ensure that:

- possible misunderstandings between interviewer and interviewee have been avoided;
- the full change history of the document is publicly available (hence changes can be viewed by anyone in the document, also information about when and by whom);
- no transcription of interviews was needed.

All input sessions were conducted on the same document, hence starting the first session with an empty template and finalizing the last with a relatively mature state. Sessions were organized in the order of self-assessed expertise level given by the makers, starting with the lowest one. It is visible in the history of the document that the last sessions only led to the change of details (specifically only in the section for PCBs then). In total, 6 sessions have been conducted, 3 with makers from the OPEN\_NEXT consortium, 3 with makers from the OSH community (specifically the Open Source Ecology Germany community).

The resulting document was commented by the team and embedded into the metadata specification (Task 3.3 of OPEN\_NEXT project). Both the resulting metadata specification and the annotated input document, were sent in their final state to all makers that took part in input sessions, requesting their feedback whether the resulting specification for production metadata meets their expectations. Feedback was collected by E-mail and processed accordingly.

### 4.2. Design notes

The specification was published under a free/open license on GitHub, both in human-readable form (in Markdown<sup>13</sup> and machine-readable form (in Turtle/RDF, as part of the ontology of T3.3<sup>14</sup>)).

---

<sup>13</sup> <https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OKH-LOSH.md#production-metadata>

<sup>14</sup> <https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OKH-LOSH.ttl>

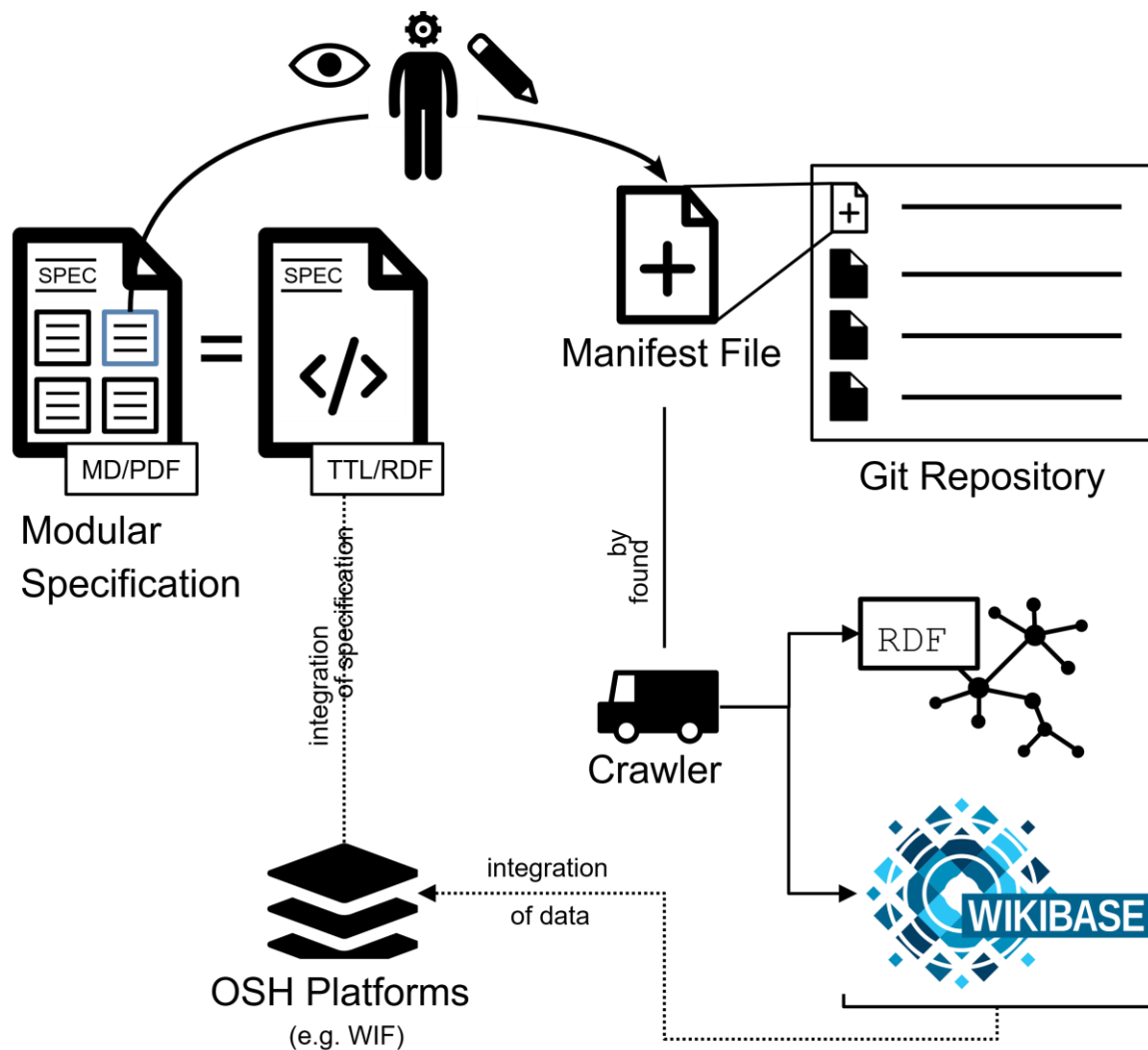


Figure 40: Provision of production metadata by publishing a plain text file („manifest file“) in a git repository

As shown in Figure 40 project team members can access the specification online, use the metadata blocks of interest and input metadata in the corresponding format into a plain text file (“manifest file”) in their GitHub repository. By that, the metadata is already publicly available. The crawler (T3.4) automatically finds and converts this data into RDF to make it available as Linked Open Data for any platform and uploads it the Wikibase instance (T3.3).

All documents were designed in a manner that makes them as freely exploitable as possible for others. Using the Markdown format for the human-readable version of the specification allows for (automated) exports in a large variety of formats (e.g. PDF, ODT, DOCX, EPUB) while being easily editable and git-compatible (since it’s a plain text format). TTL/RDF allows for referencing and integration in any context of Linked Open Data, e.g. by platform owners.

The repository can be downloaded on GitHub as a ZIP file or via the git command line tool using SSH.

However, to read or implement the specification, no installation or download is required. Data fields, properties, definitions etc. can be referenced directly from the TTL file (e.g. with the base URL <https://github.com/OPEN-NEXT/OKH-LOSH/raw/master/OKH-LOSH.ttl#> to reference the outer-dimensions property in the dimension of millimeters).

## 5. Collaborative Production – Identifying production metadata in documentation through machine learning

### 5.1. Development Method

There are various OSH platforms with different styles, information and structure, and they are all documented in a different way. As a community member searches for a specific production metadata, it can be challenging to find the right project through first glance.

In order to make this process easier and to identify some of the production metadata in documentation, it was necessary to go through all documentation and spot the specific entities. In order to identify these entities, different kind of machine learning algorithms are searched. Named Entity Recognition (NER) is used in Natural Language Processing (NLP) to extract information into pre-defined characteristics.

The algorithm uses NER of SpaCy to train the model with Deep Learning (NN) for the production metadata characteristics like manufacturing process, materials, machine type and dimensions. With a web application, any kind of text can be given as input, and the output will be provided with the classified entities. The application can be used for any plain text input or Mediawiki-based websites (such as Appropedia projects<sup>15</sup>). The recommended pre-requisites for the following sections for this solution are defined as follows and these are also present in the GitHub repository<sup>16</sup> under requirements.txt.

Required software:

- Python interpreter (version 3.7 or later)
- Doccano (1.3.0)

Required libraries:

- SpaCy >= 3.0.5
- json >= 2.0.9
- Tkinter >= 8.6
- requests >= 2.24.0
- bs4 as BeautifulSoup >= 4.9.3
- Pandas >= 1.1.3
- Pickle >= 4.0

Need to download an IDE environment on your computer, clone the repository on you IDE, then to run the application, run either runModel\_support file

This guide assumes working knowledge of Git and running Python scripts.

---

<sup>15</sup> [https://www.appropedia.org/Welcome\\_to\\_Appropedia](https://www.appropedia.org/Welcome_to_Appropedia)

<sup>16</sup> <https://github.com/OPEN-NEXT/Named-Entity-Recognition-for-extracting-Open-Source-Hardware-project-metadata/blob/main/requirements.txt>

For recreating the algorithm with a different dataset and entities, the next steps should be implemented.

### 1. Creating a dataset

- To create a dataset, the basic definitions and some open source hardware platform projects can be used. The train datasets can also be found in specific dataset platforms. If the dataset is already labelled, it can directly be implemented in step 3.
- The training dataset should be saved as txt to import in labelling tool doccano later on.

For this solution, the selected characteristics or entities are manufacturing process, materials, machine types and dimensions. Some examples from the training dataset are shown in Figure 41.

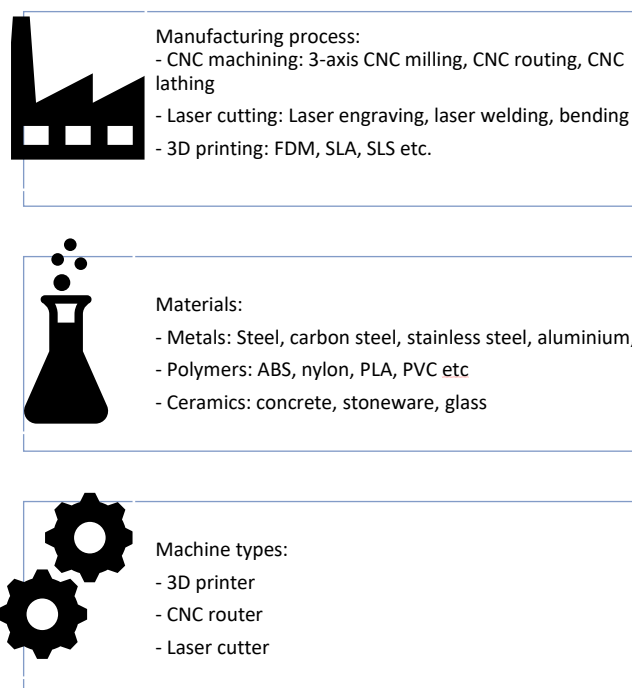


Figure 41: Example of the training data

- The complete trained dataset can be found in the GitHub repository<sup>17</sup>.

### 2. Labelling the data

This step is for labelling the entities using doccano, but if you already have labelled data, you can skip this step and directly go to training the model. Doccano<sup>18</sup> is an open source annotation tool to create

<sup>17</sup> <https://github.com/OPEN-NEXT/Named-Entity-Recognition-for-extracting-Open-Source-Hardware-project-metadata/tree/main/Raw%20Data>

<sup>18</sup> <https://github.com/doccano/doccano>

labeled data. There are different tools to label a dataset, in this solution the tool doccano is used and will be explained further.

- First step is to install doccano, please follow the doccano<sup>19</sup> instructions and open the program.
- For Windows installation,
  - pip install doccano
  - doccano
- Go to <http://127.0.0.1:8000/>
- Login with username: admin and password:password.

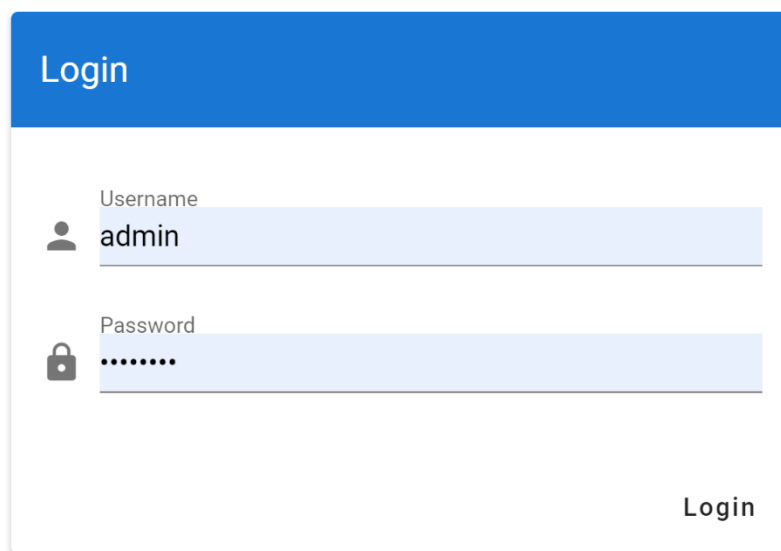


Figure 42: Login for doccano

- Click create, type in your project name, description and select the sequence labelling project type.

---

<sup>19</sup> <https://doccano.github.io/doccano/getting-started/>

**Add Project**

Project name  
Named Entity Recognition

Description  
Labeling for NER

Project type  
Sequence Labeling

Randomize document order

Share annotations across all users

Cancel Create

Figure 43: Creating a project in doccano

- After creating project, click on dataset and import your dataset.

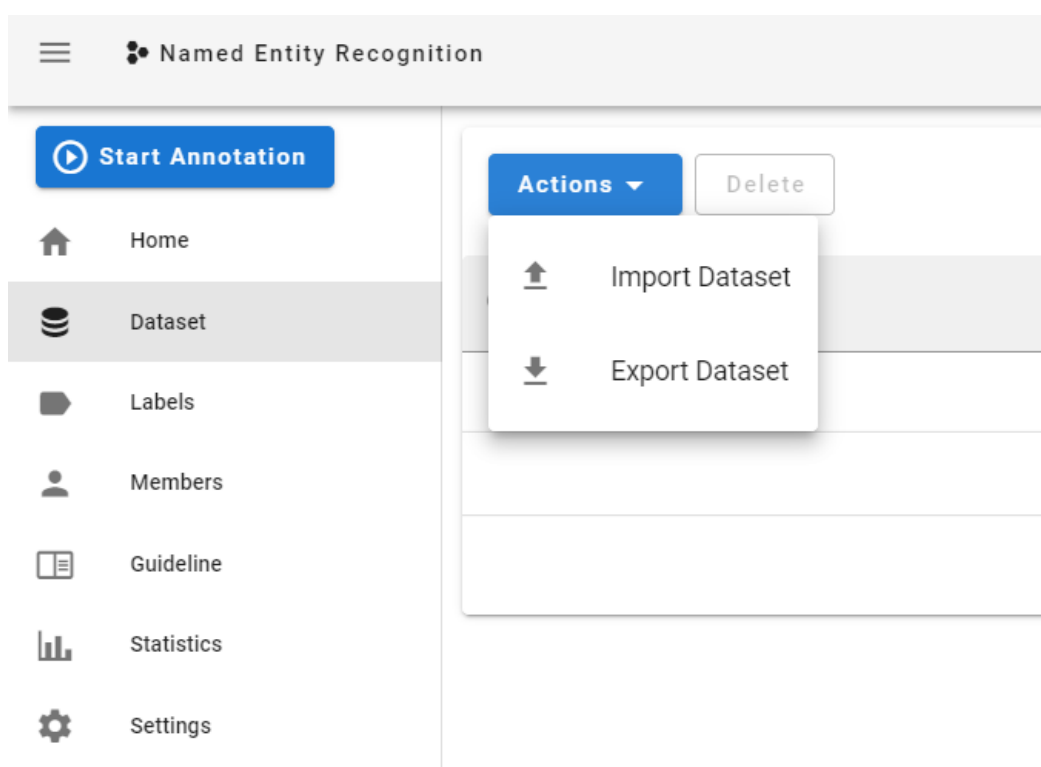


Figure 44: Importing a dataset for doccano



- Select the file format you have for your data. Make sure you have either plain text, JSONL (JavaScript Object Notation Lines) or CoNLL (Conference on Natural Language Learning) format for your unlabelled data.

### Upload Data

#### Select a file format

Plain text

JSONL

CoNLL

```
{ "text": "EU rejects German call to boycott British lamb.", "labels": [ [0, 2, "ORG"], [11, 17, "MISC"], ... ] }  
{ "text": "Peter Blackburn", "labels": [ [0, 15, "PERSON"] ] }  
{ "text": "President Obama", "labels": [ [10, 15, "PERSON"] ] }
```

#### Select file(s)

File input

File is required

[Cancel](#) [Upload](#)

- Go to Labels and create your labels.

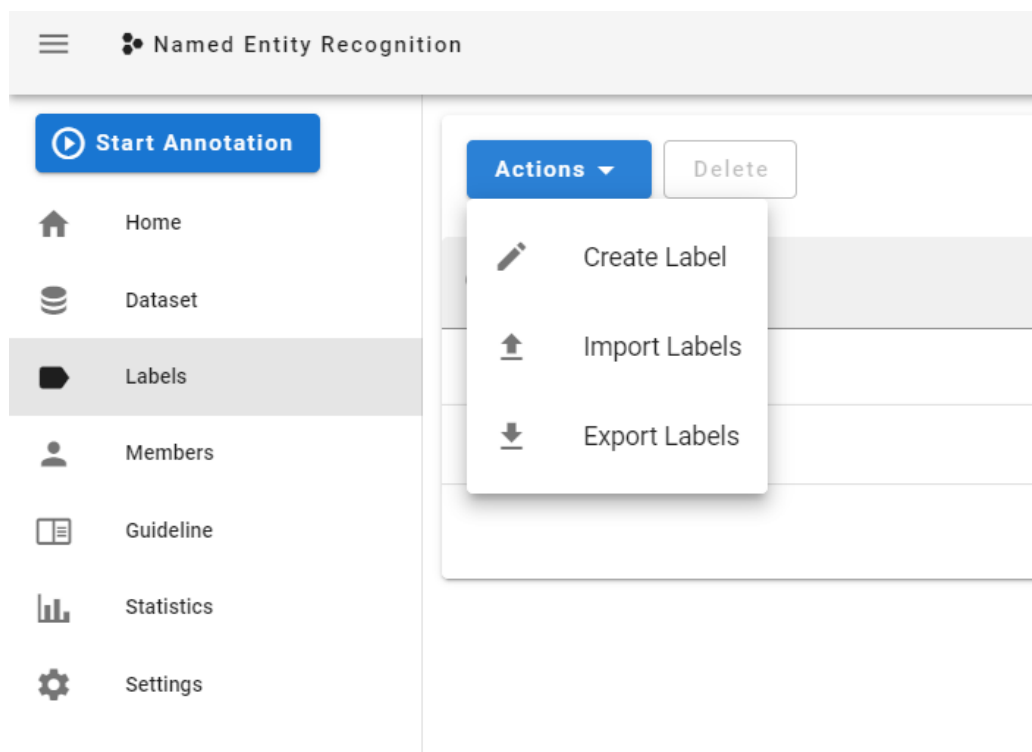


Figure 45: Creating labels in doccano

- While creating a label, make sure you enter a suitable name, a shortcut on keyword is also optional and the color.
- Go back to your imported dataset and click annotate, now you can start annotating, the dataset should look like in Figure 46.

<input type="checkbox"/>	Text	Metadata	Action
<input type="checkbox"/>	In case you have downloaded a PDO template and you want to prepare it for laser cutting, check the following post where you can find a tutorial about how to create the DXF files from a Pepakura Desig...	{}	Annotate
<input type="checkbox"/>	However, for those who don't want to use Pepakura, in this project you'll learn how to use any CAD program (Rhinceros in this case) to create the patterns from a 3D model and export them on DXF file...	{}	Annotate
<input type="checkbox"/>	Once the files have been prepared, the necessary parameters are defined in CorelDraw (in this case, the program that governs the laser cutter we used) to correctly assign the cutting and folding lines...	{}	Annotate
<input type="checkbox"/>	3D printing, or additive manufacturing, is the construction of a three-dimensional object from a CAD model or a digital 3D model	{}	Annotate
<input type="checkbox"/>	The term "3D printing" can refer to a variety of processes in which material is deposited, joined or solidified under computer control to create a three-dimensional object	{}	Annotate
<input type="checkbox"/>	with material being added together (such as plastics, liquids or powder grains being fused together), typically layer by layer.	{}	Annotate
<input type="checkbox"/>	In the 1980s, 3D printing techniques were considered suitable only for the production of functional or aesthetic prototypes, and a more appropriate term for it at the time was rapid prototyping.	{}	Annotate
<input type="checkbox"/>	As of 2019, the precision, repeatability, and material range of 3D printing have increased to the point that some 3D printing processes are considered viable as an industrial-production technology, wh...	{}	Annotate
<input type="checkbox"/>	Drilling is a cutting process that uses a drill bit to cut a hole of circular cross-section in solid materials. The drill bit is usually a rotary cutting tool, often multi-point. The bit is pressed ag...	{}	Annotate
<input type="checkbox"/>	In rock drilling, the hole is usually not made through a circular cutting motion, though the bit is usually rotated. Instead, the hole is usually made by hammering a drill bit into the hole with quick...	{}	Annotate

Figure 46: Imported dataset in doccano

- The first entity will open, you can select the word/s you want to label and continue until all your dataset is labelled accordingly
- After finishing the labelling process, click "Export Dataset" as JSONL file format under Actions and save the file.

### 3. Training the model

- First step is to install spacy
- Firstly, we read the JSONL file we exported after labelling with doccano.

```
import json
labeled_data = []
with open(r"project_1_dataset_v4.jsonl", "r", encoding='utf-8') as read_file:
    for line in read_file:
        data = json.loads(line)
        labeled_data.append(data)
print(labeled_data)
```

Figure 47: Reading labelled dataset

- After reading the data, convert the format to SpaCy format

```

TRAINING_DATA = []
for entry in labeled_data:
    entities = []
    for e in entry['labels']:
        entities.append((e[0], e[1],e[2]))
    spacy_entry = (entry['text'], {"entities": entities})
    TRAINING_DATA.append(spacy_entry)
print(TRAINING_DATA)

```

Figure 48: Converting the format

- Next step is to train the model- We use Deep Learning (NN) and set a dropout rate of 0.3 to prevent overfitting.

```

import spacy
import random
import json
from spacy.tokens import Doc
from spacy.training import Example
nlp = spacy.blank("en")
ner = nlp.create_pipe("ner")
nlp.add_pipe('ner')
for _, annotations in TRAINING_DATA:
    entities are get the name token.label_ one #goes through all the
        for ent in annotations.get("entities"):
            ner.add_label(ent[2])
# Start the training
nlp.begin_training()
# Loop for 40 iterations
for itn in range(40):
    # Shuffle the training data
    random.shuffle(TRAINING_DATA)
    losses = {}
# Batch the examples and iterate over them
    for batch in spacy.util.minibatch(TRAINING_DATA, size=2):
        for text, annotations in batch:
            # create Example
            doc = nlp.make_doc(text)
            example = Example.from_dict(doc, annotations)
            # Update the model
            nlp.update([example], losses=losses, drop=0.3)
            example = Example.from_dict(doc, annotations)
print(losses)

```

Figure 49: Training the model

- While training data, some warnings may be shown at first, then the iterations should start and take a few minutes

```
{'ner': 138.85293078339757}  
{'ner': 151.93528051267654}  
{'ner': 119.03163002112368}  
{'ner': 140.14641903853297}  
{'ner': 128.42898518155988}  
{'ner': 99.11539392433102}  
{'ner': 111.00088516619448}  
{'ner': 128.24347579402692}  
{'ner': 126.03423812205475}  
{'ner': 105.2783918711287}  
{'ner': 110.75637424712896}  
{'ner': 120.65806941707919}  
{'ner': 114.07485118403504}  
{'ner': 117.75703915068753}  
{'ner': 99.5361274841243}  
{'ner': 85.15650694469508}  
{'ner': 120.82150434878682}  
{'ner': 222.10507821918392}  
{'ner': 108.40540405753205}
```

Figure 50: Iterations through training

- After iterations stop, we save the model

```
nlp.to_disk("./my.model")
```

Figure 51: Saving the model

- Testing the model, you can write your example text in the “example” and see the results.

```
from spacy import displacy  
example = "an example test"  
doc = nlp(example)  
displacy.render(doc, style='ent')
```

Figure 52: Testing the model

## 5.2.Design notes

In conclusion, the following steps should be performed to identify metadata:

- Create a train dataset from OSH project websites.
- Label the dataset with the selected entities using doccano labelling tool manually.
- Save the labelled data as JSONL format.
- Use SpaCy Neural Network model to train a new statistical model.
- Save the model

- Then create a Spacy NLP pipeline and use the new model to detect entities corresponding your characteristics.

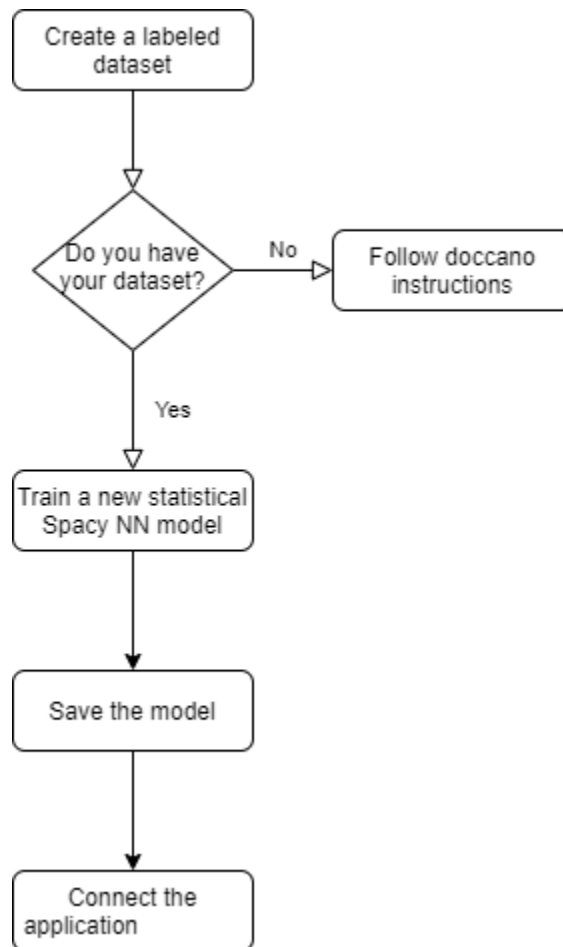


Figure 53: Flowchart of steps to recreate a NER

### 5.3. Installation guide

1. Copy the URL of the repository in Github<sup>20</sup> as in Figure 54

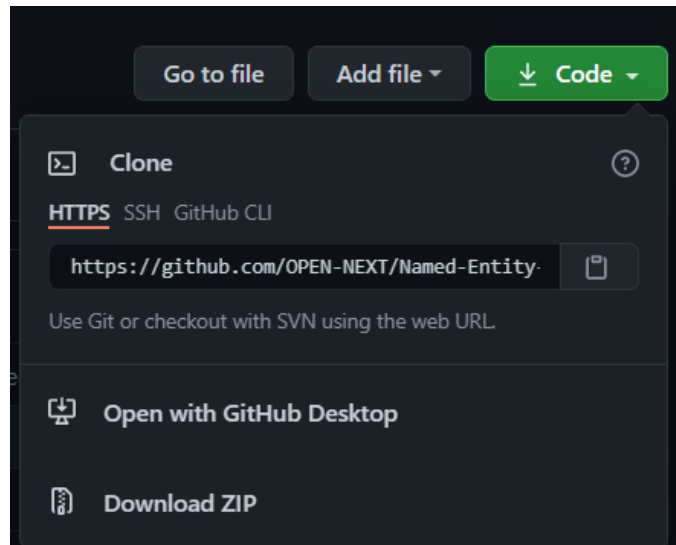


Figure 54: Copying the URL of the repository

2. Open a Python IDE (Integrated development environment) to clone the GitHub repository, in our case we used PyCharm Community Edition 2020.3.2.
3. Create a new project and save it.
4. In the project, go to VCS and click on create git repository as seen in Figure 55. If you do not have VCS option shown as in Figure 55, you need to install the package for your IDE.

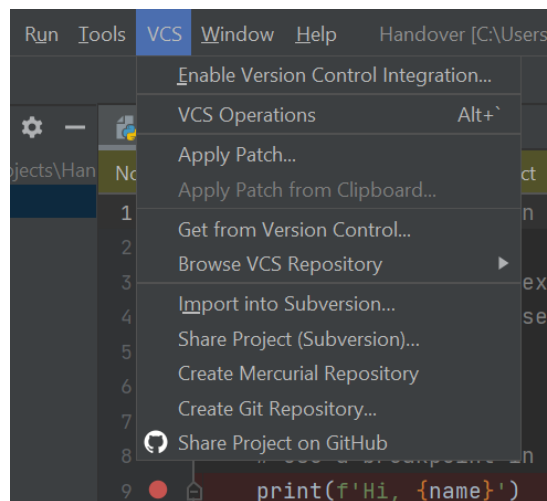


Figure 55: Create Git Repository

5. After cloning the repository successfully, it will show the tab Git instead of VCS

<sup>20</sup> <https://github.com/OPEN-NEXT/Named-Entity-Recognition-for-extracting-Open-Source-Hardware-project-metadata>

6. Under Git select clone and paste the URL you have copied in step 1.

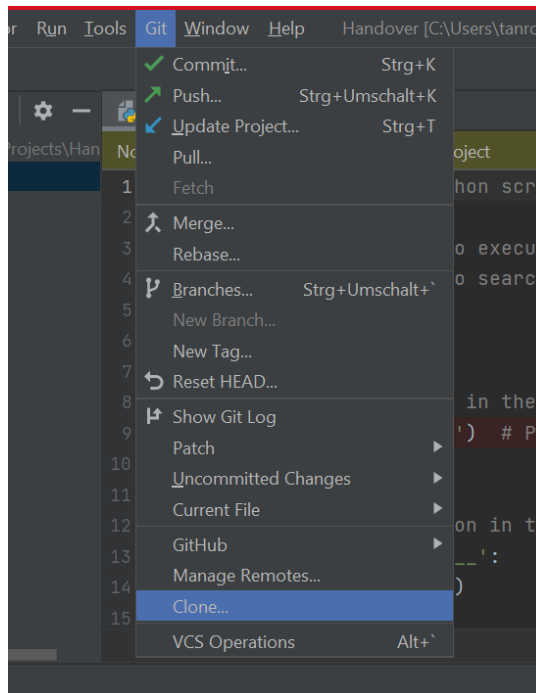


Figure 56: Clone the URL

7. After cloning the repository, it will open in your IDE, you need to create the python environment if it has not happened automatically, and the python compiler for opening application.

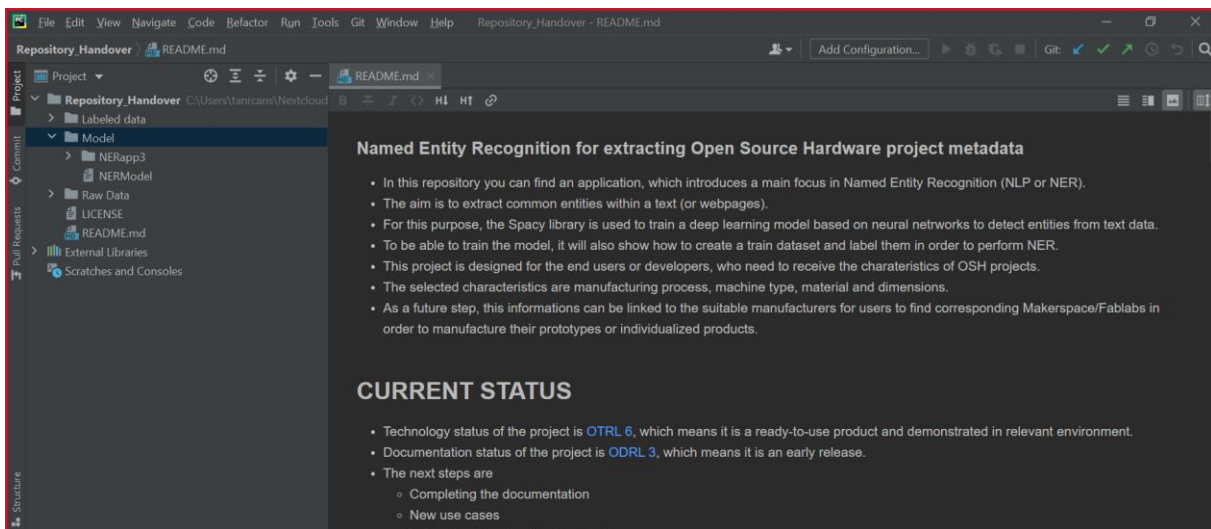


Figure 57: The cloned repository in IDE

8. Go from directory to selected app, run the runModel\_support.

9. The application will start in a new window, you can give your input and see the results.

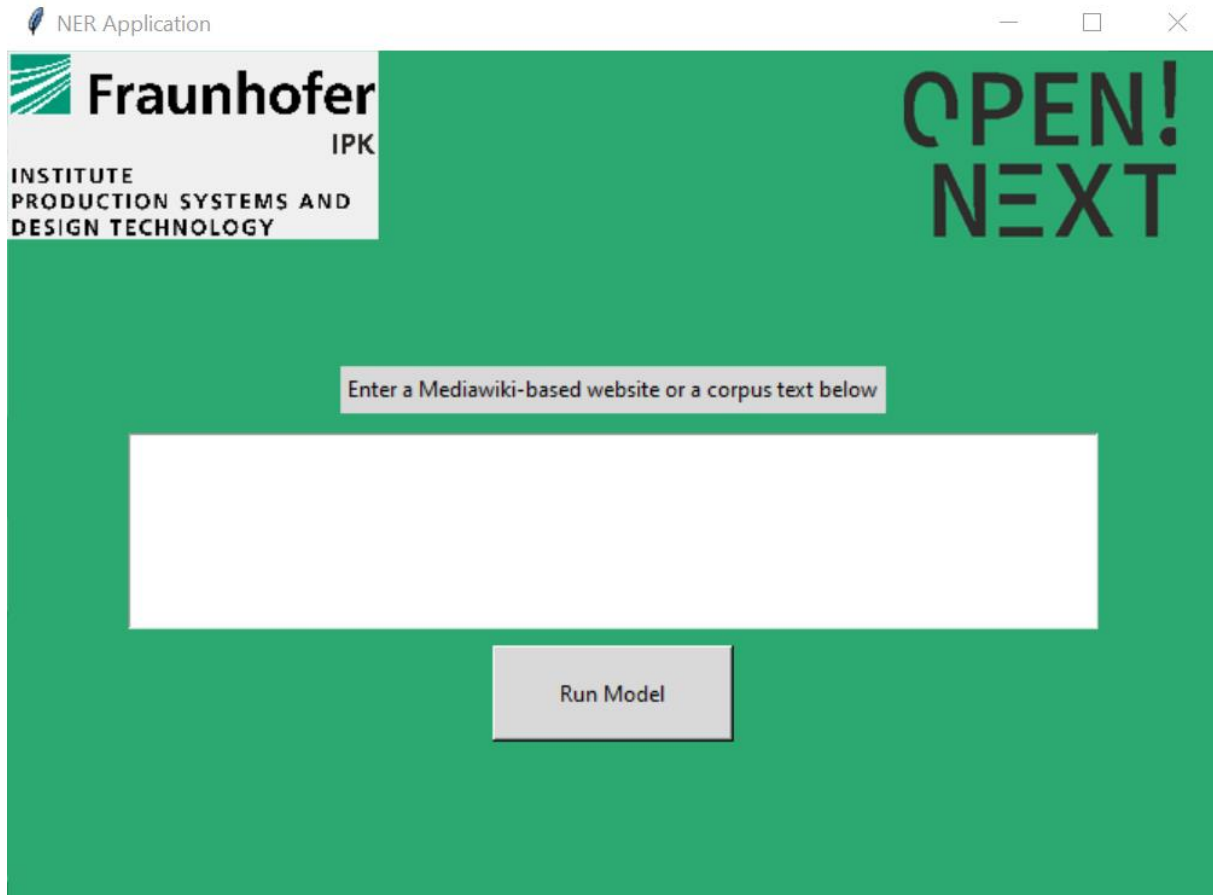


Figure 58: Front end of the application



## 6. Documentation and Guidelines – Project overview template

### 6.1. Development method and design notes

Requirements for the template were:

- to provide assistance for documenting and presenting an overview of the actual state of the project.
- to provide a total picture about the numerous project details (such as licensing, maturity of the project, motivation, next steps and many more)
- to be applicable to any platform and to be easily usable

Bases on these requirements, a commonly used template on Wikifactory<sup>21</sup> was taken as reference and further developed based on the interview results from the previous deliverable D3.1. New sections and additional questions were added to enhance the completeness of the template<sup>22</sup>. Certain sections were also deleted to avoid increasing the complexity of the project overview. Markdown was chosen as the suitable format for the template, because of the ease of use and replicability on various platforms and tools<sup>23</sup>.

## 7. Documentation and Guidelines -Technology-readiness levels for OSH

### 7.1. Development method and design notes

Requirements for the model were:

- to allow for rapid and intuitive assessment of maturity of a certain technology for a certain use case and operation environment
- to allow for easy monitoring of the progress made by the project in this regard
- to enable assessment of project for any project team member, skilled in the corresponding field(s) of technology, without training in the use of the classification model
- to be applicable for a large variety of technologies

As a result, the TRL from EU-H2020 Annex G<sup>24</sup> has been used. These in turn are based on NASA’s TRL<sup>25</sup>, which "provide useful insights into two key contributors to readiness:

1. degree of functionality provided
2. fidelity of the environment (to the intended operational environment) in which this functionality has been demonstrated"

---

<sup>21</sup> <https://wikifactory.com/+wikifactory/project-example-template/file/README.md>

<sup>22</sup> <https://github.com/OPEN-NEXT/WP3-Documentation-Guidelines-for-OSH-Projects/blob/main/Documentation%20%26%20Guidelines/Project%20overview%20documentation%20Template.md>

<sup>23</sup> <https://jaantollander.com/post/scientific-writing-with-markdown/>

<sup>24</sup> [https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014\\_2015/annexes/h2020-wp1415-annex-g-trl\\_en.pdf](https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf)

<sup>25</sup> <https://apps.dtic.mil/dtic/tr/fulltext/u2/a443149.pdf>

The TRL stated in EU-H2020 Annex G comes with a level of abstraction high enough to be applicable for very diverse technologies and are suitable to monitor the progress of a project over time<sup>26</sup>. However, the high abstraction also limits the informative value, e.g. regarding safety, usability, efficiency or maintainability of the product (just to name a few)<sup>27</sup>. Nevertheless, the TRL approach has been chosen for its well-tested application on a wide range of different hardware technologies. The yet relatively simple classification into 9 stages makes it suitable for a decentralized use in the context of open source hardware. This approach aims to foster a deeper understanding for the maturity of technological open source solutions among industrial players, research institutes and OSH Communities; contributing to the overall comparability of OSH products.

Unlike proprietary hardware, open source hardware is meant to be used, made, modified and distributed by anyone (ref: DIN SPEC 3105-1<sup>28</sup>). This in turn is made possible by 1) the license under which the technical documentation is published and 2) the quality of the technical documentation. As a result, two scales are introduced:

1. Technology Readiness Levels for the context of OSH (OTRL)
2. Documentation Readiness Levels for the context of OSH (ODRL)

For OTRL the titles of the TRL in EU-H2020 Annex G have been reused and partly shortened. Following ESA’s handbook<sup>29</sup> and NASA’s definitions<sup>30</sup> they have been mapped onto the context of OSD.

For the documentation quality, two end points of a scale have been defined:

1. Basic descriptions about the OSH product available, no replication possible (equals ODRL1)
2. Documentation allows for fully independent replication, operation and modification of the OSH product, hence fully complies with the requirements stated in DIN SPEC 3105-1 (equals ODRL5).

To keep the scale easy to handle and yet representative for important stages of the documentation quality, the scale was divided into 5 stages. In order to communicate the case of unavailable documentation, which can also be the case for mature releases when a non-open-source license is used, an additional stage (ODRL0) has been defined.

The current draft of the classification model is accessible under a free/open license on GitHub, both in human-readable form (in Markdown<sup>31</sup> and machine-readable form (in TTL/RDF, as part of the ontology of T3.3<sup>32</sup>)).

For usage or implementation, no installation or download is required. Data fields, properties, definitions etc. can be referenced directly from the TTL file (e.g. With the base URL [https://github.com/OPEN-NEXT/OKH-LOSH/raw/master/OTRL.ttl#<sup>33</sup>](https://github.com/OPEN-NEXT/OKH-LOSH/raw/master/OTRL.ttl#33) to reference OTRL5)

---

<sup>26</sup> [https://ec.europa.eu/isa2/sites/isa/files/technology\\_readiness\\_revisited\\_-\\_icegov2020.pdf](https://ec.europa.eu/isa2/sites/isa/files/technology_readiness_revisited_-_icegov2020.pdf)

<sup>27</sup> <https://apps.dtic.mil/dtic/tr/fulltext/u2/a443149.pdf>

<sup>28</sup> <https://www.beuth.de/en/technical-rule/din-spec-3105-1/324805763>

<sup>29</sup> [https://artes.esa.int/sites/default/files/TRL\\_Handbook.pdf](https://artes.esa.int/sites/default/files/TRL_Handbook.pdf)

<sup>30</sup> [https://www.nasa.gov/pdf/458490main\\_TRL\\_Definitions.pdf](https://www.nasa.gov/pdf/458490main_TRL_Definitions.pdf)

<sup>31</sup> <https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OTRL.md>

<sup>32</sup> <https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OTRL.ttl>

<sup>33</sup> <https://github.com/OPEN-NEXT/OKH-LOSH/raw/master/OTRL.ttl>

## 8. Documentation and Guidelines - Licensing open source hardware guide

### 8.1. Development method

Since licensing of OSH is a major concern among pilots in WP4/5 and based on the interview results in the previous Deliverable D3.1, the team reached out to the OSH community to find reliable material and experts willing to contribute. A cooperation was started with the legal issues working group of Open Source Ecology Germany e.V. (non-profit), which is one of the biggest community-driven organization for open source hardware in Europe. In reconciliation with their working group, the team created a compact, intuitive guideline about how IP law works for open source hardware and how licensing is applied. The knowledge was primarily derived from the extensive “OSH Guideline | Legal Issues”<sup>34</sup> distributed by their legal issues working group. The OSH community gave feedback to shape the form of the document so that other community members could read and understand it. The final document aims to provide essential legal knowledge at a very low threshold within a short amount of time.

All material for the document (including pictograms) are used under a free/open license. The document itself has been released under a free/open license on GitHub<sup>35</sup>. Hence, the full content is freely exploitable for anyone and any use.

---

<sup>34</sup> <https://gitlab.com/OSGermany/osh-guideline-legal-issues>

<sup>35</sup> <https://github.com/OPEN-NEXT/tldr-ipr>

## 9. References

Antoniou, Grigoris; van Harmelen, Frank (2009): Web Ontology Language: OWL. In Steffen Staab, Rudi Studer (Eds.): Handbook on Ontologies. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 91–110.

Ding, Ying; Jacob, Elin K.; Zhang, Zhixiong; Foo, Schubert; Yan, Erjia; George, Nicolas L.; Guo, Lijiang (2009): Perspectives on social tagging. In *J. Am. Soc. Inf. Sci.* 60 (12), pp. 2388–2401. DOI: 10.1002/asi.21190.

Mies, Robert (2021): OPEN!\_D2.1\_Open Source Product Development Process Models. DOI: 10.5281/zenodo.5341253

Pan, Jeff Z.; Vetere, Guido; Gomez-Perez, Jose Manuel; Wu, Honghan (2017): Exploiting Linked Data and Knowledge Graphs in Large Organisations. Cham: Springer International Publishing. DOI: 10.1007/978-3-319-45654-6

Treude, Christoph; Storey, Margaret-Anne (2009): How tagging helps bridge the gap between social and technical aspects in software development. In : IEEE 31<sup>st</sup> International Conference on Software Engineering, 2009. ICSE 2009 ; 16 - 24 May 2009, Vancouver, Canada ; proceedings. 2009 IEEE 31<sup>st</sup> International Conference on Software Engineering. Vancouver, BC, Canada, 5/16/2009 - 5/24/2009. Institute of Electrical and Electronics Engineers; Association for Computing Machinery; IEEE International Conference on Software Engineering; ICSE. Piscataway, NJ: IEEE, pp. 12–22.

Wang, Tao; Wang, Huaimin; Yin, Gang; Ling, Charles X.; Li, Xiao; Zou, Peng (2014): Tag recommendation for open source software. In *Front. Comput. Sci.* 8 (1), pp. 69–82. DOI: 10.1007/s11704-013-2394-x.